ON THE COMPLEXITY OF PACKAGE RECOMMENDATION PROBLEMS

TING DENG*, WENFEI FAN†, AND FLORIS GEERTS‡

Abstract. Recommendation systems aim to recommend items that are likely to be of interest to users. This paper investigates several issues fundamental to such systems.

(1) We model recommendation systems for packages (sets) of items. We use queries to specify multi-criteria for item selections and express compatibility constraints on items in a package, and use functions to compute the cost and usefulness of items to a user.

(2) We study recommendations of points of interest, to suggest top-k packages. We also investigate recommendations of top-k items, as a special case.

(3) We identify several problems, to decide whether a set of packages makes a top-k recommendation, and whether a rating bound is maximum for selecting top-k packages. We also study function problems for computing top-k packages, and counting problems to find how many packages meet the user's criteria.

(4) We establish the upper and lower bounds of these problems, *all matching*, for combined and data complexity. These results reveal the impact of variable sizes of packages, the presence of compatibility constraints, as well as a variety of query languages for specifying selection criteria and compatibility constraints, on the analyses of these problems.

Key words. Recommendation problems, Complexity

AMS subject classifications. 68P15, 03D15

1. Introduction.

Recommendation systems, *a.k.a.* recommender systems, recommendation engines or platforms, aim to identify and suggest information items or social elements that are likely to be of interest to users. Traditional recommendation systems are to select top-k items from a collection of items, *e.g.,* books, music, news, Web sites and research papers [3], which satisfy certain criteria identified for a user, and are ranked by a rating (*a.k.a.* utility) function. More recently recommendation systems are often used to find top-k packages, *i.e., sets of items*, such as travel plans [34], teams of players [20] and various course combinations [17, 24, 25]. The items in a package are required not only to meet multi-criteria for selecting individual items, but also to satisfy compatibility constraints and aggregate constraints defined on all the items in a package taken together, such as team formation [20] and course prerequisites [24]. Packages may have variable sizes subject to a cost budget, and are ranked by overall ratings of their items [34].

Recommendation systems are increasingly becoming an integral part of Web services [34], Web search [4], social networks [4], education software [25] and commerce services [3]. A number of systems have been developed for recommending items or packages, known as *points of interest (POI)* [34] (see [3, 4] for surveys). These systems use relational queries to specify selection criteria and compatibility constraints [2, 7, 17, 25, 34]. There has also been work on the complexity of computing POI recommendations [20, 24, 25, 34].

However, to understand central issues associated with recommendation systems, there is much more to be done. (1) The previous complexity results were developed for

*School of Computer Science and Engineering, Beihang University, Beijing, China (dengting@act.buaa.edu.cn).

†School of Informatics, University of Edinburgh, Edinburgh EH8 9LE, UK (wenfei@inf.ed.ac.uk).

‡Department of Mathematics and Computer Science, University of Antwerp, Antwerp, Belgium (Floris.Geerts@ua.ac.be).

individual applications with specific selection criteria and compatibility constraints. They may not carry over to other settings. This highlights the need for studying recommendation problems in a *uniform model.* (2) In most cases only lower bounds were given (NP-hard by *e.g.,* [24, 25]). Worse still, among the few upper bounds claimed, some are not quite correct. It is necessary to set the record straight by establishing *matching* upper and lower bounds. (3) No previous work has studied the precise causes for high complexity. Is it from variable sizes of packages, compatibility constraints or from complex selection criteria? The need for understanding this is evident when developing practical recommendation systems.

EXAMPLE 1.1. Consider a recommendation system for travel plans, which maintains two relations specified by

$$\mathsf{flight}(\mathsf{f\#}, \mathsf{From}, \mathsf{To}, \mathsf{DT}, \mathsf{DD}, \mathsf{AT}, \mathsf{AD}, \mathsf{Pr}) \text{ and } \mathsf{POI}(\mathsf{name}, \mathsf{city}, \mathsf{type}, \mathsf{ticket}, \mathsf{time}).$$

Here a flight tuple specifies flight f# from From to To that departs at time DT on date DD and arrives at time AT on date AD, with airfare Pr. A POI tuple specifies a place name to visit in the city, its ticket price, type (*e.g.,* museum, theater), and the amount of time needed for the visit.

(1) Recommendations of items. A user wants to find top-3 flights from Edinburgh (EDI) to New York City (NYC) with at most one stop, departing on 1/1/2012, with lowest possible airfare and duration time. This can be stated as item recommendation: (a) flights are items; (b) the selection criteria are expressed as a union $Q_1 \cup Q_2$ of conjunctive queries, where Q_1 and Q_2 select direct and one-stop flights from EDI to NYC leaving on 1/1/2012, respectively; and (c) the items selected are ranked by a rating function $f()$: given an item s, $f(s)$ is a real number computed from the airfare Pr and the duration Dur of s such that the higher the Pr and Dur are, the lower the rating of s is. Here Dur can be derived from DT, DD, AT and AD, and $f()$ may associate different weights with Pr and Dur.

(2) Recommendations of packages. One is planing a 5-day holiday, by taking a direct flight from EDI to NYC departing on 1/1/2012 and visiting as many places in NYC as possible. In addition, no more than 2 museums should be in a package, which is a compatibility constraint [34]. Moreover, plans should have the lowest overall price. This is an example of package recommendations: (a) the selection criteria are expressed as the following conjunctive query (CQ) Q, which finds pairs of flights and POI as items:

$$\begin{aligned} Q(\mathsf{f\#}, \mathsf{Pr}, \mathsf{name}, \mathsf{type}, \mathsf{ticket}, \mathsf{time}) = \exists\, \mathsf{DT}, \mathsf{AT}, \mathsf{AD} \\ \big(\mathsf{flight}(\mathsf{f\#}, \textsc{edi}, \textsc{nyc}, \mathsf{DT}, 1/1/2012, \mathsf{AT}, \mathsf{AD}, \mathsf{Pr}) \\ \wedge\ \mathsf{POI}(\mathsf{name}, \textsc{nyc}, \mathsf{type}, \mathsf{ticket}, \mathsf{time})\big); \end{aligned}$$

(b) a package N consists of some items that have the same f# (and hence Pr); (c) the rating of N, denoted by $\mathsf{val}(N)$, is a real number such that the higher the sum of the Pr and ticket prices of the items in N is, the lower $\mathsf{val}(N)$ is; (d) the compatibility constraint can be expressed as a CQ query Q_c such that $Q_c(N) = \emptyset$ if and only if no more than 2 museums are present in a package. To assert this constraint we use the following CQ query:

$$\begin{aligned} Q_c() = \exists\, \mathsf{f\#}, \mathsf{Pr}, n_1, t_1, p_1, n_2, t_2, p_2, n_3, t_3, p_3 \big(R_Q(\mathsf{f\#}, \mathsf{Pr}, n_1, \mathit{museum}, p_1, t_1) \\ \wedge\ R_Q(\mathsf{f\#}, \mathsf{Pr}, n_2, \mathit{museum}, p_2, t_2)\ \wedge R_Q(\mathsf{f\#}, \mathsf{Pr}, n_3, \mathit{museum}, p_3, t_3) \\ \wedge\ n_1 \neq n_2 \wedge\ n_1 \neq n_3 \wedge\ n_2 \neq n_3\big), \end{aligned}$$

where R_Q denotes the schema of the query answer $Q(D)$. Since $Q_c(N) \neq \emptyset$ if there are more than three distinct museums in N, $Q_c(N) = \emptyset$ asserts the desired constraint; and (e) the cost of N, denoted by $\mathsf{cost}(N)$, is the total time taken for visiting all POI in N. Note that the number of items in N is *not* fixed: N may contain as many POI as possible, as long as $\mathsf{cost}(N)$ does not exceed the total time allocated for sightseeing in 5 days. Putting these together, the travel planning is to find top-k such packages ranked by $\mathsf{val}(N)$, for a constant k chosen by the user.

(3) Computational complexity. To develop a recommendation system, one naturally wants to know the complexity for computing top-k packages or top-k items. The complexity may depend on what query language we use to specify selection criteria and compatibility constraints. For instance, in the package recommendation example given above, the criteria and constraints are expressed as CQ queries. Suppose that the user also requires that there are at least 2 museums in a package N. This cannot be expressed by CQ queries. Instead, one can consider the following Q'_c in first-order logic (FO):

$$\begin{aligned} Q'_c() = \forall\ & \mathsf{f\#}, \mathsf{Pr}, n_1, t_1, p_1, n_2, t_2, p_2 \\ & \big(R_Q(\mathsf{f\#}, \mathsf{Pr}, n_1, \mathit{museum}, p_1, t_1)\ \wedge R_Q(\mathsf{f\#}, \mathsf{Pr}, n_2, \mathit{museum}, p_2, t_2) \\ & \qquad\qquad\qquad \to (n_1 = n_2 \wedge\ p_1 = p_2 \wedge\ t_1 = t_2)\big). \end{aligned}$$

Obviously, Q'_c returns nonempty on a package N if and only if N contains at most one museum. Furthermore, suppose that the user can bear with indirect flights with an unlimited number of stops. Then we need to express the selection criteria in, *e.g.,* DATALOG, which is more costly to evaluate than the CQ queries. What is the complexity of package recommendations when criteria and constraints are expressed in various languages? Will the complexity be lower if compatibility constraints are absent? Will it make our lives easier if we fix the size of each package? To the best of our knowledge, these questions have not been answered in previous work. $\diamond$

These highlight the need for a full treatment of recommendation problems, to study them in a generic model, establish their matching upper and lower bounds, and identify where the complexity arises.

A model for package recommendations. Following [2, 7, 17, 24, 25, 34] we consider a database D that includes items in a recommendation system. We specify (a) multi-criteria for selecting items as a relational query Q; (b) compatibility constraints on the items in a package N as another query Q_c such that $Q_c(N, D) = \emptyset$ if and only if N satisfies the constraints; (c) a rating function $\mathsf{val}()$ from packages to real numbers $\mathbb{R}$ such that $\mathsf{val}(N)$ assesses the usefulness of a package N to a user; and (d) a cost budget C and a function $\mathsf{cost}()$ from packages to $\mathbb{R}$ such that a package N is a "valid" choice if and only if $\mathsf{cost}(N) \leq C$. Given a positive integer k, package recommendation is to find top-k packages based on $\mathsf{val}()$ such that each package consists of items selected by Q and satisfies the constraints Q_c. As shown in Example 1.1, packages may have *variable sizes*: we want to maximize $\mathsf{val}(N)$ as long as $\mathsf{cost}(N)$ does not exceed the budget C.

Traditional item recommendations are a special case of package recommendations. We use a rating function $f()$ to give a rating in $\mathbb{R}$ to each tuple in $Q(D)$. For a given k, item recommendation is to find top-k items that meet the criteria specified by Q, ranked by the function $f()$ (see detailed discussions in related work).

This yields a model for top-k package recommendation that subsumes previous models studied for, *e.g.,* travel and course recommendations. We study recommendation problems in a generic setting when selection criteria and compatibility constraints are expressed as queries, and when the functions $\mathsf{cost}()$, $\mathsf{val}()$ and $f()$ are only assumed to be computable in PTIME *w.r.t.* the size of packages N and tuples t, respectively.

Recommendation problems. We identify several problems for POI recommendations. (a) Decision problems RPP and MBP are to decide whether a set of packages is a top-k recommendation, and whether a constant B is the largest bound such that there exists a top-k recommendation in which each package N is rated above B, *i.e.,* $\mathsf{val}(N) \geq B$, respectively. (b) Function problem FRP is to find a top-k recommendation if there exists one. (c) Counting problem CPP counts how many valid packages exist that have ratings above a bound B.

We study the impact of various dimensions on the complexity of these problems.

(a) Query languages for specifying queries Q and compatibility constraints Q_c. We consider various query languages $\mathcal{L}_Q$ in which selection criteria Q and compatibility constraints Q_c are expressed. More specifically, the query language $\mathcal{L}_Q$ ranges over the following (see *e.g.,* [1] for details):

- conjunctive queries (CQ), built up from atomic formulas with constants and variables, *i.e.,* relation atoms in database schema $\mathcal{R}$ and built-in predicates $(=, \neq, <, \leq, >, \geq)$, by closing under conjunction $\wedge$ and existential quantification $\exists$;
- union of conjunctive queries (UCQ) of the form $Q_1 \cup \cdots \cup Q_r$, where for each $i \in [1, r]$, Q_i is in CQ;
- positive existential FO queries ($\exists$FO$^+$), built from atomic formulas by closing under $\wedge$, *disjunction* $\vee$ and $\exists$;
- nonrecursive datalog queries (DATALOG$_{\mathsf{nr}}$), defined as a collection of rules of the form $p(\vec{x}\,) \leftarrow p_1(\vec{x}_1), \ldots, p_n(\vec{x}_n)$, where the head p is an IDB predicate and each p_i is either an atomic formula or an IDB predicate, such that its dependency graph is acyclic; the *dependency graph* of a DATALOG query Q is a directed graph $G_Q = (V, E)$, where V includes all the predicates of Q, and (p', p) is an edge in E if and only if p' is a predicate that appears in a rule with p as its head [8];
- first-order logic queries (FO) built from atomic formulas using $\wedge$, $\vee$, *negation* $\neg$, $\exists$ and universal quantification $\forall$; and
- datalog queries (DATALOG), defined as a collection of rules $p(\vec{x}) \leftarrow p_1(\vec{x}_1), \ldots, p_n(\vec{x}_n)$, for which the dependency graph may possibly be cyclic, *i.e.,* DATALOG is an extension of DATALOG$_{\mathsf{nr}}$ with an inflational fixpoint operator.

(b) Size bounds on packages. We investigate the impact of only allowing packages of a fixed size versus allowing packages of variable size. Observe that item recommendations can be viewed as package recommendations in which each package consists of a single tuple only.

(c) Compatibility constraints. We compare the setting in which no compatibility constraints are present with the case where such constraints are present. In addition, we consider "simple" compatibility constraints Q_c that can be evaluated in PTIME in the sizes of Q_c and database D.

(d) The number k of packages. We study whether the complexity of the problems RPP, FRP, MBP and CPP depends on the number k of packages to be recommended. More specifically, we consider two cases: $k = 1$ (top-1 packages) and variable k (top-k packages).

(e) Finally, we also study item recommendations for which each package consists of a single item and compatibility constraints are absent.

Complexity results. For all these problems we establish their combined and data complexity [1]. In the case of combined complexity, we allow the query Q, compatibility constraints Q_c and database D to vary. For data complexity we only allow the database D to vary and regard Q and Q_c as being predefined and fixed.

These results give a complete characterization of the complexity in this model, from decision problems to function and counting problems. They tell us where complexity arises, complementing previously stated results.

(a) Query languages dominate the complexity of recommendation problems, *e.g.,* the problem for deciding the maximum bound for top-k package recommendations ranges from $\mathsf{D}_2^{\mathsf{p}}$-complete for CQ, PSPACE-complete for FO and DATALOG$_{\mathsf{nr}}$, to $\mathsf{EXPTIME}$-complete for DATALOG.

(b) Variable package sizes do not make our lives harder when combined complexity is concerned for all the languages given above. Indeed, when packages may have variable sizes, all these problems have the same combined complexity as their counterparts when packages are restricted to be singleton sets. In fact, variable sizes of packages have impact only on data complexity, or when $\mathcal{L}_Q$ is a simple language with a PTIME complexity for its membership problem (*i.e.,* given a query $Q \in \mathcal{L}_Q$, a database D and a tuple t, to decide whether $t \in Q(D)$). These clarify the impact of package sizes studied in, *e.g.,* [34].

(c) The presence of compatibility constraints does not increase the combined complexity when the query language $\mathcal{L}_Q$ is FO, DATALOG$_{\mathsf{nr}}$ or DATALOG. Indeed, for these languages, all the problems for package recommendations and their counterparts for item recommendations have the same complexity. Furthermore, these constraints also do *not* complicate the data complexity analyses. However, compatibility constraints increase combined complexity when $\mathcal{L}_Q$ is contained in $\exists\text{FO}^+$. In the absence of compatibility constraints, the decision problem for top-k package recommendations is DP-complete and its function problem is $\mathsf{FP}^{\mathsf{NP}}$-complete when $\mathcal{L}_Q$ is CQ. They are DP-hard and $\mathsf{FP}^{\mathsf{NP}}$-hard, respectively, even when selection criteria are given by an identity query. In contrast, when compatibility constraints are present, these problems become Π_2^p-complete and $\mathsf{FP}^{\Sigma_2^p}$-complete, respectively, when $\mathcal{L}_Q$ is CQ. These give precise bounds for the problems studied in, *e.g.,* [34].

(d) The choice of k does not have an impact on the complexity. Indeed, recommending top-1 packages (resp. items) is as hard as recommending top-k-packages (resp. items).

(e) The complexity of item recommendation problems coincides with the complexity of their package recommendation counterparts in the absence of compatibility constraints (for combined complexity) and when packages are restricted to be of fixed size (for data complexity).

These results are also of interest to the study of top-k query answering, among other things. A variety of techniques are used to prove our results, including a wide range of reductions, and constructive proofs with algorithms (*e.g.,* for the function problems). In particular, the proofs demonstrate that the complexity of these problems for CQ, UCQ and $\exists\text{FO}^+$is inherent to top-k package querying itself, rather than a consequence of the complexity of the query languages.

Related work. Traditional recommendation systems aim to find, for each user, items that maximize the user's rating function (see, *e.g.,* [3] for a survey). Selection criteria are decided by content-based, collaborative and hybrid approaches, which consider preferences of each user in isolation, or preferences of similar users [3]. There has been a large body of interesting work on item recommendation, mostly focusing on how to choose appropriate rating functions, and how to extrapolate such functions when they are not defined on the entire item space, by deriving unknown values from known ones. Historical user behaviors and transaction data (logs) are typically considered for extrapolation. Orthogonal to the prior work, we focus on the computational complexity of recommendation problems in this work. To this end, we *assume a given rating function* that is total, *i.e.,* they are defined on the entire item space, and study where the complexity arises when computing top-k items. We adopt a simple model that supports content-based, collaborative and hybrid criteria in terms of various queries, and assumes that historical user behaviors and transaction data are collected in the database and can be employed in queries for selecting items.

Recently, recommendation systems have been extended to finding packages, which are presented to the user in a ranked order based on some rating function [6, 20, 24, 25, 34]. A number of algorithms have been developed for recommending packages of a fixed size [6, 20] or variable sizes [24, 25, 34]. Compatibility constraints [20, 24, 25, 34] and budget restrictions [34] on packages have also been studied. Instead of considering domain-specific applications, we model recommendations of both items and packages (fixed size or polynomial size) by specifying general selection criteria and compatibility constraints as queries, and supporting aggregate constraints defined in terms of cost budgets and rating bounds.

Several decision problems for course package recommendations have been shown to be NP-hard [24, 25]. It was claimed that the problems of forming a team with compatibility constraints [20] and the problem of finding packages that satisfy some budget restrictions (without compatibility constraints) [34] are NP-complete. In contrast, we establish the precise complexity of a variety of problems associated with POI recommendations (Table 7.1 and 7.2, Section 7).

There has also been a host of work on recommending items and packages taken from views of the data [2, 7, 17, 21, 34]. Such views are expressed as relational queries, representing preferences or points of interest [2, 7, 17]. Here recommendations often correspond to top-k query answers. Indeed, top-k query answering retrieves the k-items (tuples) from a query result that are top-ranked by some rating function [16]. Such queries either simply select tuples, or join and aggregate multiple inputs to find top-k tuples, by possibly incorporating user preference information [17, 28]. A number of top-k query evaluation algorithms have been developed (*e.g.,*[13, 21, 27]; see [16] for a survey), as well as algorithms for incremental computation of ranked query results [9, 14, 22] that retrieve the top-k query answers one at a time. A central issue there concerns how to combine different ratings of the same item based on multiple criteria. Our work also retrieves tuples from the result of a query. It differs from the previous work in the following. (1) In contrast to top-k query answering, we are to find items and sets of items (packages) provided that a rating function is given. (2) We focus on the complexity of recommendations problems rather than the efficiency or optimization of query evaluation.

It should be remarked that real-life recommendation systems typically do not expect a normal user to write complexity queries and constraints. The systems aim to provide their users with useful information, interface, or a fixed set of query forms so that the users can easily specify their selection criteria and compatibility constraints. In this context, the data complexity analyses of recommendation problems (Section 5) are applicable. Furthermore, the combined complexity results may help recommendation system vendors or developers decide what query languages and compatibility constraints to support, by striking a balance between the expressive power needed and the cost (complexity) introduced.

This paper extends [11] by including detailed proofs which can be found in the appendix. To keep the paper within a reasonable page limit we do not consider query relaxation and adjustment recommendation, both of which are studied in [11].

Organization. The remainder of the paper is organized as follows. Section 2 introduces the model for package recommendations. Section 3 formulates four fundamental problems RPP, FRP, MBP and CPP in connection with POI recommendations. We investigate the combined complexity of these four problems in Section 4, and establish their data complexity in Section 5. A variety of special cases of these problems are studied in Section 6. Section 7 summarizes the main results of the paper and identifies open issues. Detailed proofs are deferred to the Appendix.

2. Modeling recommendations. In this section we present our model for recommendations of packages and items in detail.

Item collections. Following [2, 7, 17, 24, 25, 34], we assume a database D consisting of items for selection, possibly along with historical user behaviors for rating the items. The database is specified with a relational schema $\mathcal{R}$ composed of a collection of relation schemas $(R_1, \ldots, R_n)$. Each schema R_i is defined over a fixed set of attributes. For each attribute A in R_i, its domain is specified in R_i and is denoted by $\mathsf{dom}(R_i.A)$.

Package recommendations. As remarked earlier, in practice one often wants packages of items, *e.g.,* combinations of courses to be taken to satisfy the requirements for a degree [25], travel plans including multiple POI [34], and teams of experts [20]. Package recommendation is to find top-k packages such that the items in each package (a) meet the selection criteria, (b) satisfy some compatibility constraints, *i.e.,* they have no conflicts, and moreover, (c) their ratings and costs satisfy certain aggregate constraints. To specify these, we extend the models proposed in [25, 34] as follows.

Selection criteria. We use a query Q in a query language $\mathcal{L}_Q$ to specify multi-criteria for selecting items from D. For instance, as shown in Example 1.1, we use a query to specify what flights and sites a user wants to find.

A *package* is a subset $N \subseteq Q(D)$. We use R_Q to denote the relation schema of the query result $Q(D)$, which can be easily derived from query Q and the schema $\mathcal{R}$ of D. To simplify the discussion, we assume *w.l.o.g.* two predefined polynomial p and p_s such that (1) the number $|N|$ of items in N is no larger than $p(|D|)$, where $|D|$ is the size of D, and (2) the arity of R_Q (*i.e.,* the number of attributes in the tuples of N) does not exceed $p_s(|Q|, |\mathcal{R}|)$, where $|Q|$ is the size of query Q, and $|\mathcal{R}|$ is the size of schema $\mathcal{R}$. Indeed, it is not of much practical use to find items of exponentially large or a package with exponentially many items.

Compatibility constraints. To specify the compatibility constraints for a package N, we use a query Q_c such that N satisfies Q_c if and only if $Q_c(N, D) = \emptyset$. That is, Q_c identifies inconsistencies among items in N. To simplify the discussion, we assume that query Q_c for specifying compatibility constraints and query Q for specifying selection criteria are in the same language $\mathcal{L}_Q$. If a system supports compatibility constraints in $\mathcal{L}_Q$, there is no reason for not supporting queries in the same language for selecting items. We defer to future work the study in the setting when Q_c and Q are expressed in different languages. Note that queries in various query languages are capable of expressing compatibility constraints commonly found in practice, including those studied in [17, 20, 24, 25, 34].

Aggregate constraints. To specify aggregate constraints, we define a cost function and a rating function over packages, following [34]: (1) $\mathsf{cost}(N)$ computes a value in $\mathbb{R}$ as the cost of package N; and (2) $\mathsf{val}(N)$ computes a value in $\mathbb{R}$ as the overall rating of N. For instance, $\mathsf{cost}(N)$ in Example 1.1 is computed from the total time taken for visiting POI, while $\mathsf{val}(N)$ is defined in terms of airfare and total ticket prices. We also assume a cost budget C, and specify an aggregate constraint $\mathsf{cost}(N) \leq C$. For instance, the cost budget C in Example 1.1 is the total time allowed for visiting POI in 5 days, and the aggregate constraint $\mathsf{cost}(N) \leq C$ imposes a bound on the number of POI in a package N.

We just assume that $\mathsf{cost}()$ and $\mathsf{val}()$ are PTIME computable aggregate functions *w.r.t.* the size of packages N, defined in terms of *e.g.,* max, min, sum, avg, as commonly found in practice.

Top-k package selections. For a database D, queries Q and Q_c in $\mathcal{L}_Q$, a natural number $k \geq 1$, a cost budget C, and functions $\mathsf{cost}()$ and $\mathsf{val}()$, a *top-k package selection* is a set $\mathcal{N} = \{N_i \mid i \in [1, k]\}$ of packages such that for each $i \in [1, k]$,

(1) $N_i \subseteq Q(D)$, *i.e.,* its items meet the criteria given in Q;
(2) $Q_c(N_i, D) = \emptyset$, *i.e.,* the items in the package satisfy the compatibility constraints specified by query Q_c;
(3) $\mathsf{cost}(N_i) \leq C$, *i.e.,* its cost is below the budget;
(4) for all packages $N' \notin \mathcal{N}$ that satisfy conditions (1–3) given above, $\mathsf{val}(N') \leq \mathsf{val}(N_i)$, *i.e.,* packages in $\mathcal{N}$ have the k highest overall ratings among all feasible packages; and
(5) $N_i \neq N_j$ if $i \neq j$, *i.e.,* the packages are pairwise distinct.

Note that packages in $\mathcal{N}$ may have *variable* sizes. That is, the number of items in each package is not necessarily bounded by a constant. We just require that N_i satisfies the constraint $\mathsf{cost}(N_i) \leq C$, $|N_i|$ does not exceed a predefined polynomial p in $|D|$, and that the arity of tuples in N_i is bounded by a predefined polynomial p_s in $|Q|$ and $|\mathcal{R}|$. In Section 6, we will also consider a fixed size bound for $|N_i|$.

The *package recommendation problem* is to find a *top-k package selection* for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$, if there exists one. As shown in Example 1.1, users may want to find, *e.g.,* a top-k travel-plan selection with the minimum price. Note that there may exist more than one top-k packages. In this case, we assume that there is either a strategy in place to break the ties, *e.g.,* based on some ordering on packages, or that a single top-k package is non-deterministically selected.

As advocated in [2, 7, 17, 24, 25, 34], selection criteria, compatibility constraints, and aggregate constraints for rating are key components for specifying package recommendations. Along the same lines, we treat these parameters separately in our model to explore the impact of each factor on the complexity of package recommendation.

Item recommendations. To rank items, we use a *rating function* $f()$ to measure the usefulness of items selected by $Q(D)$ to a user [3]. It is a PTIME-computable function that takes a tuple s from $Q(D)$ and returns a real number $f(s)$ as the rating of item s. The functions may incorporate users' preferences [28] and historical behaviors, and may be *different for different users*.

Given a constant $k \geq 1$, a *top-k selection for* (Q, D, f) is a set $S = \{s_i \mid i \in [1, k]\}$ such that

(a) $S \subseteq Q(D)$, *i.e.,* items in S satisfy the criteria specified by Q;
(b) for all $s \in Q(D) \setminus S$ and $i \in [1, k]$, $f(s) \leq f(s_i)$, *i.e.,* items in S have the highest ratings; and
(c) $s_i \neq s_j$ if $i \neq j$, *i.e.,* items in S are *distinct*.

Given D, Q, f and k, the *item recommendation problem* is to find a top-k selection for (Q, D, f) if there exists one. For instance, a top-3 item selection is described in Example 1.1, where items are flights and the rating function $f()$ is defined in terms of the airfare and duration of each flight.

The connection between item and package selections. In our model item selections are a special case of package selections. Indeed, a top-k selection $S = \{s_i \mid i \in [1, k]\}$ for (Q, D, f) is a top-k package selection $\mathcal{N}$ for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$, where $\mathcal{N} = \{N_i \mid i \in [1, k]\}$, and for each $i \in [1, k]$, (a) $N_i = \{s_i\}$, (b) Q_c is a constant query that returns $\emptyset$ on any input, referred to as the empty query; (c) $\mathsf{cost}(N_i) = |N_i|$ if $N_i \neq \emptyset$, and $\mathsf{cost}(\emptyset) = \infty$; that is, $\mathsf{cost}(N_i)$ counts the number of items in N_i if $N_i \neq \emptyset$, and the empty set is not taken as a recommendation; (d) the cost budget $C = 1$, and hence, N_i consists of a single item as imposed by $\mathsf{cost}(N_i) \leq C$; and (e) $\mathsf{val}(N_i) = f(s_i)$.

In the sequel, we use top-k package selection *specified in terms of* (Q, D, f) to refer to a top-k selection S for (Q, D, f), *i.e.,* a top-k package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$ in which $Q_c, \mathsf{cost}(), \mathsf{val}()$ and C are defined as above.

We say that compatibility constraints are *absent* if Q_c is the empty query; *e.g.,* Q_c is absent in item selections.

Examples. As a case study, below we show how course package recommendation [24] and team formation [20] can be specified in our model.

Example 2.1. The course package recommendation problem [24] is to find a top-1 package of a fixed number of courses such that some prerequisite constraints are satisfied, *i.e.,* if a course is recommended then all its prerequisites must also be recommended as well, such that the package maximizes some rating function $\mathsf{val}()$. We can easily specify course recommendation in our model as follows. The system maintains a database D_{course} consisting of two relations specified by $\mathsf{course}(\mathsf{cid}, \mathsf{title})$ and $\mathsf{prereq}(\mathsf{cid}_1, \mathsf{cid}_2)$. Here a course tuple specifies course title identified by cid. A prereq tuple encodes that course cid_1 is a prerequisite for course cid_2. Packages are selected by the identity query Q_{id} on the course relation. Since we are interested in packages N of a fixed size m, we define $\mathsf{cost}(N) = \infty$ if $|N| \neq m$ and $\mathsf{cost}(N) = |N|$ otherwise. We set $C = m$ such that only packages of size m will be recommended. Moreover, the prerequisite requirement [24] can be expressed as a compatibility constraint:

$$Q_{\mathsf{prereq}} = \exists\, c, n, c', n' \big(R_Q(c', n') \wedge \mathsf{course}(c, n) \wedge \mathsf{prereq}(c, c') \wedge \neg R_Q(c, n)\big).$$

Note that this query (compatibility constraint) needs to access not only courses in N but also the prerequisite relation stored in the database. Clearly, a top-1 package selection for $(Q_{\mathsf{id}}, D_{\mathsf{course}}, Q_{\mathsf{prereq}}, \mathsf{cost}(), \mathsf{val}(), C)$ corresponds to recommending a course package as described in [24].

As another example, consider the team formation problem of [20]. Assume a number of available experts X, each specified with a set of skills. There is also collaboration graph G that specifies how the experts collaborate with each other.

The team formation problem is then to find a set $X' \subseteq X$ such that the joint set of skills of those experts in X' suffices to perform a given task T and furthermore, X' has a minimal *communication cost*. Two kinds of costs are put forward in [20]: (a) the weight of the longest shortest path between any two nodes in graph G'; and (b) the sum of the weights of the edges in a minimum spanning tree on G'. Here G' is the subgraph of G induced by X'.

This problem can be readily formulated in our model. Indeed, let D_{exp} be the database storing G, Q_{id} the identity query, and let Q_{subg} be a query for a compatibility constraint such that $Q_{\mathsf{sung}}(N) = \emptyset$ if N is a subgraph of G that is induced by the vertices occurring in N. Furthermore, for a team N of experts, we define $\mathsf{cost}(N)$ to be the number of required skills for a task that are not exhibited by any other expert in N, while $\mathsf{val}(N)$ is defined in terms of the *communication cost* of N such that the higher the communication cost of N is, the lower $\mathsf{val}(N)$ is. Clearly, $\mathsf{cost}()$ and $\mathsf{val}()$ are PTIME computable functions in the size of the teams. Finally, we set the cost budget C to be 0 such that $\mathsf{cost}(N) \leq C$ ensures that the experts in team N meet the skill requirements of the task. It can then be readily verified that a top-1 package selection for $(Q_{\mathsf{id}}, D_{\mathsf{exp}}, Q_{\mathsf{subg}}, \mathsf{cost}(), \mathsf{val}(), C)$ corresponds to recommending a team of experts as stated in [20]. $\diamond$

3. Recommendations of POI's. We investigate four problems for package recommendations, namely, $\mathsf{RPP}(\mathcal{L}_Q)$, $\mathsf{FRP}(\mathcal{L}_Q)$, $\mathsf{MBP}(\mathcal{L}_Q)$ and $\mathsf{CPP}(\mathcal{L}_Q)$, stated as follows, which are fundamental to computing package recommendations.

We start with a decision problem for package selections. Consider a database D of schema $\mathcal{R}$, queries Q and Q_c in a query language $\mathcal{L}_Q$, functions $\mathsf{val}()$ and $\mathsf{cost}()$, a cost budget C, and a natural number $k \geq 1$. Given a set $\mathcal{N}$ consisting of k packages, the problem is to decide whether $\mathcal{N}$ makes a top-k package selection. That is, each package N in $\mathcal{N}$ satisfies the selection criteria Q, compatibility constraint Q_c, and aggregate constraints $\mathsf{cost}(N) \leq C$ and $\mathsf{val}(N) \geq \mathsf{val}(N')$ for all $N' \notin \mathcal{N}$. As remarked earlier, we assume two predefined polynomials p and p_s such that $|N| \leq p(|D|)$ and the arity of tuples in N does not exceed $p_s(|Q|, |\mathcal{R}|)$ (omitted from the problem statement below for simplicity). Intuitively, this problem is to decide whether a set $\mathcal{N}$ of packages should be recommended.

$\mathsf{RPP}(\mathcal{L}_Q)$:	*The recommendation problem (packages).*
INPUT:	A database D, two queries Q and Q_c in $\mathcal{L}_Q$, two functions $\mathsf{cost}()$ and $\mathsf{val}()$, natural numbers C and $k \geq 1$, and a set $\mathcal{N} = \{N_i \mid i \in [1, k]\}$.
QUESTION:	Is $\mathcal{N}$ a top-k package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$?

Recommendation systems have to compute top-k packages, rather than expecting that candidate recommendations are already in place. This highlights the need for studying the function problem below, to compute top-k packages.

$\mathsf{FRP}(\mathcal{L}_Q)$:	*The function recommendation problem (packages).*
INPUT:	A database D, two queries Q and Q_c in $\mathcal{L}_Q$, two functions $\mathsf{cost}()$ and $\mathsf{val}()$, and natural numbers C and $k \geq 1$.
OUTPUT:	A top-k package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$ if it exists.

The next question concerns how to find a maximum rating bound for computing top-k packages. We say that a constant B is a *rating bound* for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(),$

$C, k)$ if (a) there exists a top-k package selection $\mathcal{N} = \{N_i \mid i \in [1, k]\}$ for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$ and moreover, (b) $\mathsf{val}(N_i) \geq B$ for each $i \in [1, k]$. That is, B allows a top-k package selection. We say that B is *the maximum bound for packages* with $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, k)$ if for all rating bounds B', $B \geq B'$. Obviously B is unique if it exists. Intuitively, when B is identified, we can capitalize on B to compute top-rated packages. Furthermore, vendors could decide, *e.g.,* the price for certain items on sale with such a bound, for risk assessment.

$\mathsf{MBP}(\mathcal{L}_Q)$:	*The maximum bound problem (packages).*
INPUT:	A database D, two queries Q and Q_c in $\mathcal{L}_Q$, two functions $\mathsf{cost}()$ and $\mathsf{val}()$, and natural numbers C, $k \geq 1$, and and B.
QUESTION:	Is B the maximum bound for packages with $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, k)$?

The last problem we consider is to count the number of valid packages. Recall that a package N is *valid* for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B)$ if (a) $N \subseteq Q(D)$, (b) $Q_c(N, D) = \emptyset$, (c) $\mathsf{cost}(N) \leq C$, and (d) $\mathsf{val}(N) \geq B$, where $|N|$ is bounded by a predefined polynomial p in $|D|$. Given B, one naturally wants to know how many valid packages are out there, and hence, can be selected. This suggests that we study the following counting problem.

$\mathsf{CPP}(\mathcal{L}_Q)$:	*The counting problem (packages).*
INPUT:	A database D, two queries Q and Q_c in $\mathcal{L}_Q$, two functions $\mathsf{cost}()$ and $\mathsf{val}()$, and natural numbers C and B.
OUTPUT:	The number of packages that are valid for (Q, D, Q_c, $\mathsf{cost}()$, $\mathsf{val}()$, C, B).

That is, $\mathsf{CPP}(\mathcal{L}_Q)$ is to count the number of valid packages satisfying the users' requirements. An effective counting procedure obviously finds applications in practice. For instance, it helps managers of recommendation systems to find out how many packages carried by the system meet the users' need, and adjust the stock accordingly.

In the rest of the paper we investigate these problems. We provide their combined complexity bounds in Section 4, data complexity bounds in Section 5, followed by complexity in various special cases in Section 6.

4. Combined complexity.

In this section, we establish the combined complexity of $\mathsf{RPP}(\mathcal{L}_Q)$, $\mathsf{FRP}(\mathcal{L}_Q)$, $\mathsf{MBP}(\mathcal{L}_Q)$ and $\mathsf{CPP}(\mathcal{L}_Q)$, when $\mathcal{L}_Q$ ranges over CQ, UCQ, $\exists$FO$^+$, FO, DATALOG$_{\mathsf{nr}}$ and DATALOG. Detailed proofs of the results in this section can be found in Appendix A.

Deciding package selections. We first consider the combined complexity of $\mathsf{RPP}(\mathcal{L}_Q)$ in the presence of compatibility constraints Q_c (Theorem 4.1). We then study the complexity of $\mathsf{RPP}(\mathcal{L}_Q)$ in the absence of Q_c (Theorem 4.2)

The result below tells us that the combined complexity of $\mathsf{RPP}(\mathcal{L}_Q)$ is mostly determined by what query language $\mathcal{L}_Q$ we use to specify selection criteria and compatibility constraints. Indeed, it is Π_2^p-complete when $\mathcal{L}_Q$ is CQ, PSPACE-complete for DATALOG$_{\mathsf{nr}}$ and FO, and it becomes $\mathsf{EXPTIME}$-complete when $\mathcal{L}_Q$ is DATALOG.

THEOREM 4.1. *For* $\mathsf{RPP}(\mathcal{L}_Q)$*, the combined complexity is*

- Π_2^p*-complete when* $\mathcal{L}_Q$ *is CQ, UCQ, or* $\exists$*FO*$^+$*;*
- PSPACE*-complete when* $\mathcal{L}_Q$ *is DATALOG*$_{\mathsf{nr}}$ *or FO; and*

- EXPTIME-*complete when* $\mathcal{L}_Q$ *is DATALOG.* □

Proof sketch. (1) We show that RPP($\mathcal{L}_Q$) is Π_2^p-hard when $\mathcal{L}_Q$ is CQ, by reduction from the complement of *the compatibility problem.* Given a query Q, a database D, compatibility constraints Q_c, two functions cost() and val(), and constants C and B, the compatibility problem is to decide whether there exists a nonempty set $N \subseteq Q(D)$ such that $Q_c(N, D) = \emptyset$, $\mathsf{cost}(N) \leq C$ and $\mathsf{val}(N) > B$. We show that the compatibility problem is Σ_2^p-complete for CQ by reduction from the $\exists^*\forall^*$3DNF problem. The latter problem is to decide whether a given sentence $\varphi = \exists X \forall Y\ \psi(X, Y)$ is true, where ψ is a disjunction $C_1 \vee \cdots \vee C_r$ and each clause C_i is a conjunction of three literals defined in terms of variables in $X \cup Y$. The $\exists^*\forall^*$3DNF problem is known to be Σ_2^p-complete [29].

For the upper bound, we give a Π_2^p algorithm for RPP($\exists$FO$^+$), which first tests, given a query Q, a database D, compatibility constraints Q_c, two functions cost() and val(), constants C, and a set $\mathcal{N}$ of packages, whether $\mathcal{N}$ is a valid package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$ in DP; it then checks whether there exists *no* package with a higher rating than some $N \in \mathcal{N}$, in Π_2^p. Here DP is the class of languages recognized in deterministic polynomial time with two calls to an oracle: one call to an NP oracle and one call to a coNP oracle. That is, L is in DP if there exists languages $L_1 \in$ NP and $L_2 \in$ coNP such that $L = L_1 \cup L_2$ (see the details about DP in, *e.g.,* [23]).

(2) We show that RPP(DATALOG$_{\mathsf{nr}}$) and RPP(FO) are PSPACE-hard by reduction from the membership problem for DATALOG$_{\mathsf{nr}}$ and FO, respectively. The membership problem is to determine, given a query Q in DATALOG$_{\mathsf{nr}}$ or FO, a database D and a tuple t, whether $t \in Q(D)$. It is know that this problem is PSPACE-complete for queries in DATALOG$_{\mathsf{nr}}$ [32] and FO [31] (see also [10] for a general survey of complexity results of the membership problem for various query languages). For the upper bound, we provide an NPSPACE algorithm to check RPP for these two languages. Thus RPP(DATALOG$_{\mathsf{nr}}$) and RPP(FO) are both in PSPACE since NPSPACE =PSPACE by Savitch's theorem [26].

(3) For DATALOG, we show that RPP is EXPTIME-hard by reduction from the membership problem for DATALOG, which is EXPTIME-complete [31]. For the upper bound, we give an EXPTIME algorithm to check RPP(DATALOG). □

One might think that the absence of compatibility constraints Q_c would make our lives easier. Indeed, RPP(CQ) becomes DP-complete in the absence of Q_c, as opposed to Π_2^p-complete in the presence of Q_c. However, when $\mathcal{L}_Q$ is powerful enough to express FO or DATALOG$_{\mathsf{nr}}$ queries, dropping Q_c does not help: RPP($\mathcal{L}_Q$) in this case has the same complexity as its counterpart when Q_c is present.

THEOREM 4.2. *In the absence of* Q_c*, the combined complexity of* RPP($\mathcal{L}_Q$) *is*
- DP-*complete when* $\mathcal{L}_Q$ *is CQ, UCQ, or* $\exists$*FO*$^+$*;*
- PSPACE-*complete when* $\mathcal{L}_Q$ *is DATALOG*$_{\mathsf{nr}}$ *or FO; and*
- EXPTIME-*complete when* $\mathcal{L}_Q$ *is DATALOG.* □

Proof sketch. (1) We show that RPP(CQ) is DP-hard by reduction from SAT-UNSAT, which is known to be DP-complete (cf. [23]). An instance of SAT-UNSAT is a pair (φ_1, φ_2) of 3SAT instances over variables in X and Y, respectively, where φ_1 is an instance $\varphi = C_1 \wedge \cdots \wedge C_r$ in which each clause C_i is a disjunction of three variables or negations thereof taken from X; similarly for φ_2 and Y. Given such a pair (φ_1, φ_2), SAT-UNSAT is to decide whether φ_1 is satisfiable and φ_2 is not satisfiable. In the reduction we define compatibility constraints Q_c as the empty query, *i.e.,* compatibility constraints are absent.

For the upper bound, consider the algorithm outlined earlier for RPP($\exists$FO$^+$) in the proof of Theorem 4.1. In the absence of Q_c, the algorithm is in DP, and hence so is RPP($\exists$FO$^+$).

(2-3) The lower bound proofs (2-3) of Theorem 4.1 for RPP(DATALOG$_{\text{nr}}$), RPP(FO) and RPP(DATALOG) do not use compatibility constraints, and hence remain intact here. The upper bounds given there obviously carry over to this special case. ☐

Computing top-k packages. To state the complexity of FRP we first recall the following complexity classes: FP$^{\text{NP}}$ is the class of all functions from strings to strings that can be computed by a PTIME Turing machine with an NP oracle (cf. [23]), and FP$^{\Sigma_2^p}$ is the class of all functions computable by a PTIME 2-alternating max-min Turing machine [18]. By FPSPACE(poly) (resp. FEXPTIME(poly)) we mean the class of all functions associated with a two-argument predicate R_L that satisfies the following conditions:

- R_L is *polynomially balanced*, *i.e.,* there is a polynomial q such that for all strings x and y, if $R_L(x, y)$ then $|y| \leq q(|x|)$, and
- the decision problem "given x and y, whether $R_L(x, y)$" is in PSPACE (resp. EXPTIME) [19].

Given a string x, the *function problem* associated with R_L is to find a string y such that $R_L(x, y)$ if such a string exists.

The complexity of the function problem FRP($\mathcal{L}_Q$) is as follows:

THEOREM 4.3. *For* FRP($\mathcal{L}_Q$)*, the combined complexity is*

- FP$^{\Sigma_2^p}$*-complete when $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$;*
- FPSPACE(poly)*-complete if $\mathcal{L}_Q$ is DATALOG$_{\text{nr}}$ or FO; and*
- FEXPTIME(poly)*-complete when $\mathcal{L}_Q$ is DATALOG.*

In the absence of compatibility constraints, its combined complexity remains unchanged for DATALOG$_{\text{nr}}$, FO and DATALOG, but it becomes FP$^{\text{NP}}$*-complete for CQ, UCQ and $\exists$FO$^+$.* ☐

These results tell us that it is nontrivial to find top-k packages. Indeed, to express compatibility constraints on travel plans given in [34], we need at least CQ; for the course combination constraints of [17, 24, 25], we need FO; and for connectivity of flights we need DATALOG. These place FRP in FP$^{\Sigma_2^p}$, FPSPACE(poly) and FEXPTIME(poly), respectively.

It was claimed in several earlier papers that when $k = 1$, it is NP-complete to find a top-1 package. Unfortunately, it is not the case. Indeed, the proofs of Theorems 4.3, and 4.1 tell us that when $k = 1$, the function problem FRP($\mathcal{L}_Q$) remains FP$^{\Sigma_2^p}$-complete and the decision problem RPP($\mathcal{L}_Q$) is Π_2^p-complete even when $\mathcal{L}_Q$ is CQ, not to mention more expressive $\mathcal{L}_Q$. Furthermore, we will show in the next section that even when Q and Q_c are both fixed, FRP is FP$^{\text{NP}}$-complete (Theorem 5.2) and RPP is coNP-complete (Theorem 5.1) when $k = 1$.

Observe that in the absence of compatibility constraints, only the analyses of the combined complexity of FRP for CQ, UCQ and $\exists$FO$^+$are simplified. This is consistent with Theorem 4.2.

Proof sketch. (1) We show that $\mathsf{FRP}(\text{CQ})$ is $\mathsf{FP}^{\Sigma_2^p}$-hard by reduction from the MAXIMUM Σ_2^p problem, which is $\mathsf{FP}^{\Sigma_2^p}$-complete [18]. The latter is to find, given a formula $\varphi(X) = \forall Y \psi(X, Y)$, the truth assignment μ_X^{last} of X that satisfies φ and comes last in the lexicographical ordering if it exists, where $\psi(X, Y)$ is an instance of 3DNF over variables in $X \cup Y$. That is, ψ is a disjunction $C_1 \vee \cdots \vee C_r$, where each clause C_i is a conjunction of three literals over $X \cup Y$.

For the upper bound, we give an $\mathsf{FP}^{\Sigma_2^p}$ algorithm (*i.e.,* an algorithm runs in polynomial time with access to a Σ_2^p-oracle) for $\mathsf{FRP}(\exists\text{FO}^+)$ that given as input Q, D, Q_c, $\mathsf{cost}()$, $\mathsf{val}()$, C and k, computes a top-k package selection if it exists.

(2) When $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ or FO (resp. DATALOG), we show that $\mathsf{FRP}(\mathcal{L}_Q)$ is $\mathsf{FPSPACE(poly)}$-hard (resp. $\mathsf{FEXPTIME(poly)}$-hard) by reducing to it all functions computable by a PSPACE (resp. $\mathsf{EXPTIME}$) Turing machine in which the output on the working tape is bounded by a polynomial. For the upper bounds, we give an algorithm in $\mathsf{FPSPACE(poly)}$ (resp. $\mathsf{FEXPTIME(poly)}$) for $\mathsf{FRP}(\mathcal{L}_Q)$ when $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ or FO (resp. DATALOG). The algorithm is a variation of the algorithm given for $\mathsf{FRP}(\exists\text{FO}^+)$ in which we replace the Σ_2^p-oracle by a PSPACE oracle (resp. $\mathsf{EXPTIME}$ oracle) when $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ or FO (resp. DATALOG).

(3) When Q_c is absent, we show that $\mathsf{FRP}(\text{CQ})$ is $\mathsf{FP}^{\mathsf{NP}}$-hard by reduction from MAX-WEIGHT SAT, which is known to be $\mathsf{FP}^{\mathsf{NP}}$-complete (cf. [23]). Given a set $\mathcal{C}$ of clauses $\{C_1, \ldots, C_r\}$ with weights, where each clause C_i is a disjunction of three literals over X, MAX-WEIGHT SAT is to find a truth assignment that satisfies a set of clauses in $\mathcal{C}$ with the most total weight. For the upper bound, the algorithm for $\mathsf{FRP}(\exists\text{FO}^+)$ given in proof (1) is now in $\mathsf{FP}^{\mathsf{NP}}$. The proofs for DATALOG$_{\mathsf{nr}}$, FO and DATALOG given in (2) still work in this special case, as no Q_c is used there when verifying the lower bounds. □

Deciding the maximum bound. We show that $\mathsf{MBP}(\text{CQ})$ is $\mathsf{D_2^p}$-complete. Here $\mathsf{D_2^p}$ is the class of languages recognized by oracle machines that make a call to an Σ_2^p oracle and a call to a Π_2^p oracle. That is, L is in $\mathsf{D_2^p}$ if there exist languages $L_1 \in \Sigma_2^p$ and $L_2 \in \Pi_2^p$ such that $L = L_1 \cap L_2$ [33], analogous to how DP is defined with NP and coNP [23].

When $\mathcal{L}_Q$ is FO, DATALOG$_{\mathsf{nr}}$ or DATALOG, $\mathsf{MBP}(\mathcal{L}_Q)$ and $\mathsf{RPP}(\mathcal{L}_Q)$ have the same complexity. Moreover, the absence of Q_c has the same impact on $\mathsf{MBP}(\mathcal{L}_Q)$ as on $\mathsf{RPP}(\mathcal{L}_Q)$.

THEOREM 4.4. *For* $\mathsf{MBP}(\mathcal{L}_Q)$*, the combined complexity is*

- $\mathsf{D_2^p}$*-complete when* $\mathcal{L}_Q$ *is CQ, UCQ or* $\exists$*FO*$^+$*;*
- PSPACE*-complete when* $\mathcal{L}_Q$ *is DATALOG*$_{\mathsf{nr}}$ *or FO; and*
- $\mathsf{EXPTIME}$*-complete when* $\mathcal{L}_Q$ *is DATALOG.*

When compatibility constraints are absent, its combined complexity remains unchanged for DATALOG$_{\mathsf{nr}}$*, FO and DATALOG, but it becomes* DP*-complete for CQ, UCQ and* $\exists$*FO*$^+$*.* □

Proof sketch. (1) We show that $\mathsf{MBP}(\text{CQ})$ is $\mathsf{D_2^p}$-hard by reduction from $\exists^*\forall^*$3DNF–$\forall^*\exists^*$3CNF, which is known to be $\mathsf{D_2^p}$-complete [33]. Given a pair (φ_1, φ_2) of $\exists^*\forall^*$3DNF instances (see the proof of Theorem 4.1 for the description of $\exists^*\forall^*$3DNF instances), the latter is to decide whether φ_1 is true and φ_2 is false. For the upper bound, we show that $\mathsf{MBP}(\exists\text{FO}^+)$ is in $\mathsf{D_2^p}$ by giving an polynomial time algorithm that makes one call to a Σ_2^p oracle and one call to a Π_2^p oracle.

(2) We show that $\mathsf{MBP}(\text{DATALOG}_{\mathsf{nr}})$, $\mathsf{MBP}(\text{FO})$ and $\mathsf{MBP}(\text{DATALOG})$ are PSPACE-hard, PSPACE-hard and EXPTIME-hard, respectively, by reduction from the membership problems of $\text{DATALOG}_{\mathsf{nr}}$, FO and DATALOG queries, respectively (see the proof of Theorem 4.1 for the details of the membership problem). The reductions are extensions of their counterparts in the proofs of Theorem 4.1 (2-3) for $\mathsf{RPP}(\text{DATALOG}_{\mathsf{nr}})$, $\mathsf{RPP}(\text{FO})$ and $\mathsf{RPP}(\text{DATALOG})$, respectively, in which compatibility constraints are all defined as the empty query. For the upper bounds, we give an NPSPACE (=PSPACE) algorithm for $\mathsf{RPP}(\mathcal{L}_Q)$ when $\mathcal{L}_Q$ is $\text{DATALOG}_{\mathsf{nr}}$ or FO, and an EXPTIME algorithm for $\mathsf{MBP}(\text{DATALOG})$.

(3) In the absence of Q_c, we show that $\mathsf{MBP}(\text{CQ})$ is DP-hard by reduction from SAT-UNSAT (see the proof of Theorem 4.2 for the details of SAT-UNSAT). For the upper bound, we show that $\mathsf{MBP}(\exists\text{FO}^+)$ is in DP by giving a polynomial time algorithm that makes one call to an NP oracle and one call to a coNP oracle. When $\mathcal{L}_Q$ is FO, $\text{DATALOG}_{\mathsf{nr}}$ or DATALOG, the lower bound proofs given in (2) above do not use compatibility constraints, and thus the lower bounds remain intact. Obviously, the upper bounds given there carry over to the special case in the absence of Q_c. □

Counting recommendations We provide the complexity of $\mathsf{CPP}(\mathcal{L}_Q)$ as follows.

THEOREM 4.5. *For $\mathsf{CPP}(\mathcal{L}_Q)$, the combined complexity is*
- $\#\cdot\mathsf{coNP}$*-complete when $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$;*
- $\#\cdot\mathsf{PSPACE}$*-complete when $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ or FO; and*
- $\#\cdot\mathsf{EXPTIME}$*-complete when $\mathcal{L}_Q$ is DATALOG.*

In the absence of compatibility constraints, its combined complexity remains unchanged for DATALOG$_{\mathsf{nr}}$, FO and DATALOG, but it becomes $\#\cdot\mathsf{NP}$-complete for CQ, UCQ and $\exists$FO$^+$. □

Here we use the framework of predicate-based counting classes introduced in [15]. For a complexity class C of decision problems, $\#\cdot\mathsf{C}$ is the class of all counting problems associated with a binary predicate R_L that satisfies the following conditions:
- R_L is polynomially balanced, *i.e.,* $R_L(x, y)$ implies $|y| \leqslant |x|^k$ for some $k \geqslant 1$; that is, the length of the second component is always bounded by a polynomial in the length of the first; and
- the decision problem "given x and y, whether $R_L(x, y)$" is in C.

A *counting problem* is to compute the cardinality of the set $\{y | R_L(x, y)\}$, *i.e.,* it is to find how many y there are such that $R_L(x, y)$ holds.

It is known that $\#\cdot\mathsf{P} = \#\mathsf{P}$, $\#\cdot\mathsf{NP} \subseteq \#\mathsf{NP} = \# \cdot \mathsf{P}^{\mathsf{NP}} = \#\cdot\mathsf{coNP}$, but $\#\cdot\mathsf{NP} = \#\cdot\mathsf{coNP}$ if and only if $\mathsf{NP} = \mathsf{coNP}$, where $\#\mathsf{P}$ and $\#\mathsf{NP}$ are counting classes in the machine-based framework of [30]. These and Theorem 4.5 tell us that the combined complexity of $\mathsf{CPP}(\text{CQ})$ is $\#\mathsf{NP}$-complete.

Proof sketch. (1) We show that $\mathsf{CPP}(\text{CQ})$ is $\#\cdot\mathsf{coNP}$-hard by reduction from $\#\Pi_1$SAT, which is $\#\cdot\mathsf{coNP}$-complete [12]. Given a universally quantified Boolean formula of the form $\varphi(X, Y) = \forall X\, \psi(X, Y)$, where ψ is a 3DNF over variables in $X \cup Y$, $\#\Pi_1$SAT is to count the number of truth assignments of Y that satisfy φ. The reduction is an 1-1 mapping from the solutions to $\mathsf{CPP}(\text{CQ})$ to the truth assignments of Y that satisfy φ, and thus is *parsimonious.* We also show that $\mathsf{CPP}(\exists\text{FO}^+)$ is in $\#\cdot\mathsf{coNP}$.

(2) We show that CPP is $\#\cdot\mathsf{PSPACE}$-hard for $\text{DATALOG}_{\mathsf{nr}}$ and FO by parsimonious reductions from #QBF, which is known to be $\#\cdot\mathsf{PSPACE}$-complete [19]. Given $\varphi =$

$\exists X \, \forall y_1 P_2 y_2 \cdots \; P_n y_n \, \psi$, where ψ is a 3SAT instance over X and $\{y_i \mid i \in [1, n]\}$, and P_i is $\forall$ or $\exists$, #QBF is to count the number of truth assignments of X that satisfy φ. For DATALOG, we verify that CPP is #·EXPTIME-hard by parsimonious reduction from all counting problems in #·EXPTIME. We also show that CPP is in #·PSPACE (resp. #·EXPTIME) for DATALOG$_{\mathsf{nr}}$ and FO (resp. DATALOG).

(3) When Q_c is absent, we show that CPP(CQ) is #·NP-hard by parsimonious reduction from #Σ_1SAT, which is #·NP-complete [12]. Given $\varphi(X, Y) = \exists X \; \psi(X, Y) = \exists X (C_1 \wedge \cdots \wedge C_r)$, where C_i's are disjunctions of variables or negated variables taken from $X \cup Y$, #Σ_1SAT is to count truth assignments of Y that satisfy φ. We also show that CPP($\exists$FO$^+$) is in #·NP. When $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$, FO or DATALOG, the proofs of (2) given above carry over since those lower bound proofs do not use compatibility constraints. □

5. Data Complexity. In practice, one often has to deal with a predefined set of queries. That is, the queries are fixed, and only the underlying databases vary. For instance, the queries given in Example 1.1 can be issued by using fixed Web forms provided by the recommendation system's Web site. This highlights the need for studying the data complexity of package recommendations, for a fixed set of queries.

In this section, we investigate the data complexity of problems RPP($\mathcal{L}_Q$), FRP($\mathcal{L}_Q$), MBP($\mathcal{L}_Q$) and CPP($\mathcal{L}_Q$), when queries Q and compatibility constraints Q_c are predefined and fixed, while database D may vary.

The complexity results given below tell us that fixing selection criteria Q and compatibility constraints Q_c do make our lives easier: RPP($\mathcal{L}_Q$), FRP($\mathcal{L}_Q$), MBP($\mathcal{L}_Q$) and CPP($\mathcal{L}_Q$) are coNP-complete, FP$^{\mathsf{NP}}$-complete, DP-complete and #·P-complete, respectively, for all the languages in Section 1. However, dropping compatibility constraints Q_c does not further reduce the complexity when data complexity is concerned. That is, all the results given in this section remain unchanged in the absence of Q_C. Detailed proofs of the results in this section can be found in Appendix B.

We start with RPP($\mathcal{L}_Q$), the recommendation problem for packages.

Theorem 5.1. *For* RPP($\mathcal{L}_Q$)*, the data complexity is* coNP*-complete when* $\mathcal{L}_Q$ *is CQ, UCQ,* $\exists$*FO*$^+$*, DATALOG*$_{\mathsf{nr}}$*, FO or DATALOG, in the presence or absence of compatibility constraints.* □

Proof sketch. We first show that the data complexity of the compatibility problem is NP-complete in the absence of compatibility constraints Q_c, by reduction from 3SAT (see the proof of Theorem 4.1 for the statement of the compatibility problem). Given an instance $\varphi = C_1 \wedge \cdots \wedge C_r$ in which each clause C_i is a disjunction of three variables or negations thereof taken from X, 3SAT is to decide whether φ is satisfiable and is known to be NP-complete (cf. [23]). From this it follows that RPP(CQ) is already coNP-hard in the absence of Q_c, by reduction from the complement of the compatibility problem. For the upper bound, we give a coNP algorithm for RPP when Q and Q_c are fixed queries in either FO or DATALOG, where Q_c may be present. □

When it comes to FRP($\mathcal{L}_Q$), the function recommendation problem, we show that fixing Q and Q_c also makes the problem easier: it drops to FP$^{\mathsf{NP}}$-complete for all the languages in Section 1.

Theorem 5.2. *The data complexity of* FRP($\mathcal{L}_Q$) *is* FP$^{\mathsf{NP}}$*-complete for CQ, UCQ,* $\exists$*FO*$^+$*, DATALOG*$_{\mathsf{nr}}$*, FO and DATALOG, in the presence or absence of compatibility constraints.* □

Proof sketch. When Q_c is absent, $\mathsf{FRP}(\text{CQ})$ is $\mathsf{FP}^{\mathsf{NP}}$-hard by Theorem 4.3. Moreover, the lower bound proof of Theorem 4.3 uses a fixed query Q in CQ (*i.e.,* a fixed identity query). Thus $\mathsf{FRP}(\text{CQ})$ is $\mathsf{FP}^{\mathsf{NP}}$-hard when Q is fixed and Q_c is absent. For the upper bound, we give an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathsf{FRP}(\text{FO})$ and $\mathsf{FRP}(\text{DATALOG})$ when Q and Q_c are fixed. ☐

We next study the maximum bound problem $\mathsf{MBP}(\mathcal{L}_Q)$. The results below tell us that fixing Q and Q_c simplifies the analysis of $\mathsf{MBP}(\mathcal{L}_Q)$.

THEOREM 5.3. *For* $\mathsf{MBP}(\mathcal{L}_Q)$*, the data complexity is* DP*-complete when* $\mathcal{L}_Q$ *is CQ, UCQ,* $\exists$*FO*$^+$*, DATALOG*$_{\mathsf{nr}}$*, FO or DATALOG, in the presence or absence of compatibility constraints.* ☐

Proof sketch. When Q_c is absent, $\mathsf{MBP}(\text{CQ})$ is DP-hard by Theorem 4.4. More, its lower bound proof uses a fixed query Q in CQ. Thus $\mathsf{MBP}(\text{CQ})$ is DP-hard when Q is fixed and Q_c is absent. For the upper bound, we give an DP algorithm for $\mathsf{MBP}(\text{FO})$ and $\mathsf{MBP}(\text{DATALOG})$ when Q and Q_c are fixed. ☐

Finally, we study the data complexity of $\mathsf{CPP}(\mathcal{L}_Q)$. The result below tells us that the data complexity of $\mathsf{CPP}(\mathcal{L}_Q)$ is $\#\mathsf{P}$-complete for all the languages considered, since $\#\mathsf{P} = \#\cdot\mathsf{P}$ (cf. [23]).

THEOREM 5.4. *For* $\mathsf{CPP}(\mathcal{L}_Q)$*, the data complexity is* $\#\cdot\mathsf{P}$*-complete when* $\mathcal{L}_Q$ *is CQ, UCQ,* $\exists$*FO*$^+$*, DATALOG*$_{\mathsf{nr}}$*, FO or DATALOG, in the presence or absence of compatibility constraints.* ☐

Proof sketch. When Q and Q_c are fixed, we show that $\mathsf{CPP}(\text{CQ})$ is $\#\cdot\mathsf{P}$-hard by parsimonious reduction from #SAT, which is $\#\cdot\mathsf{P}$-complete (cf. [23], by $\#\mathsf{P} = \#\cdot\mathsf{P}$). Given an instance ψ of 3SAT, #SAT is to count truth assignments that satisfy ψ. Furthermore, we define compatibility constraints Q_c as the empty query in the reduction. Thus $\mathsf{CPP}(\text{CQ})$ is $\#\cdot\mathsf{P}$-hard when Q is fixed and Q_c is absent. For the upper bound, we show that CPP is in $\#\cdot\mathsf{P}$ when Q and Q_c are fixed queries in FO or DATALOG, where Q_c may be present. ☐

6. Special cases of POI recommendations. The results established in the previous sections tell us that RPP, FRP, MBP and CPP have rather high complexity. In this section we revisit these problems for special cases of package recommendations, to explore the impact of various parameters of these problems on their complexity. We consider the settings when packages are bounded by a constant instead of a polynomial, when $\mathcal{L}_Q$ is a language for which the membership problem is in PTIME, and when compatibility constraints are simply PTIME functions. We also study item recommendations, for which each package has a single item, and compatibility constraints are absent. Finally, we observe that all results carry over when considering top-1 recommendations instead of top-k recommendations. Our main conclusion of this section is that the complexity bounds of these problem are rather *robust*: these restrictions simplify the analyses, but not much. Detailed proofs of the results in this section can be found in Appendix C.

6.1. Packages with a fixed bound. One might be tempted to think that fixing package size would simplify the analyses. Below we study the impact of fixing package sizes on package selections, in the presence of compatibility constraints Q_c, by considering packages N such that $|N| \leq B_p$, where B_p is a predefined *constant* rather than a polynomial.

We show that fixing package sizes does not make our lives easier when combined complexity is concerned. In contrast, it does simplify the analyses of data complexity.

COROLLARY 6.1. *For packages with a constant bound B_p, the combined complexity bounds of* RPP, FRP, MBP *and* CPP *are the same as given in Theorems 4.1, 4.3, 4.4 and 4.5, respectively; and the data complexity is*

- *in* PTIME *for* RPP,
- *in* FP *for* FRP,
- *in* PTIME *for* MBP, *and*
- *in* FP *for* CPP,

when $\mathcal{L}_Q$ is CQ, UCQ, $\exists$FO$^+$, DATALOG$_{\mathsf{nr}}$, FO or DATALOG. The complexity remains unchanged even when B_p is fixed to be 1. □

Proof sketch. (1) For the combined complexity, the lower bounds of RPP, FRP, MBP and CPP in the presence of Q_c hold here, since their proofs given in Theorems 4.1, 4.3, 4.4 and 4.5, respectively, use only top-1 package with one item. Moreover, all the upper bounds given there carry over to this special case

(2) For fixed Q and Q_c, we give algorithms in PTIME, FP, PTIME and FP for RPP, FRP, MBP and CPP, respectively. □

6.2. SP queries. In contrast, for queries that have a PTIME complexity for their membership problem, variable package sizes lead to higher complexity of RPP, FRP, MBP and CPP than their counterparts for packages with a fixed bound.

To illustrate this, we consider SP queries, a simple fragment of CQ queries that support projection and selection operators only. An SP query is of the form

$$Q(\vec{x}\,) = \exists \vec{x},\, \vec{y}\ (R(\vec{x}, \vec{y}\,) \wedge \psi(\vec{x}, \vec{y}\,)),$$

where ψ is a conjunction of predicates $=, \neq, <, \leq, >$ and $\geq$.

The result below holds for all query languages with a PTIME membership problem, including but not limited to SP. In fact the lower bounds remain intact even when the selection criteria are specified by an *identity query*, when $|\vec{y}\,| = 0$ and ψ is a tautology.

COROLLARY 6.2. *For SP queries, the combined complexity and data complexity are*

- coNP-*complete for* RPP, *but in* PTIME *for packages with a fixed (constant) bound B_p;*
- FP$^{\mathsf{NP}}$-*complete for* FRP, *but in* FP *for fixed B_p;*
- DP-*complete for* MBP, *but in* PTIME *for fixed B_p; and*
- #·P-*complete for* CPP, *but in* FP *for fixed B_p.*

when compatibility constraints are present or absent. □

Proof sketch. (1) For packages of variable sizes, the lower bounds of RPP, FRP, MBP and CPP with fixed Q in CQ hold for SP. Indeed, their proofs for Theorems 5.1, 5.2, 5.3 and 5.4 use an identity query as Q, which is in SP. For the upper bounds, the algorithms given there for RPP, FRP, MBP and CPP with a fixed Q apply to arbitrary SP queries.

(2) For packages with a constant bound, the algorithms for fixed Q of Corollary 6.1 apply to SP queries, fixed or not. □

6.3. PTIME **compatibility constraints.** One might also think that we would get lower complexity with PTIME compatibility constraints. That is, we simply treat compatibility constraints as PTIME functions in the sizes of $|Q_c|$, $|D|$ and $|N|$ of compatibility constraints Q_c, database D and packages N, respectively, rather than queries in $\mathcal{L}_Q$. Nonetheless, in this setting, the complexity remains the same as its counterpart when Q_c is absent, no better and no worse.

Corollary 6.3. *With* PTIME *compatibility constraints* Q_c*,*

- *the combined complexity of* RPP*,* FRP MBP *and* CPP *remains the same as their counterparts in the absence of* Q_c*, as given in Theorems 4.2, 4.3, 4.4 and 4.5, respectively, and*
- *their data complexity remains the same as their counterparts in the absence of* Q_c*, as given in Theorems 5.1, 5.2, 5.3 and 5.4, respectively,*

when $\mathcal{L}_Q$ *is CQ, UCQ,* $\exists$*FO*$^+$*, DATALOG*$_{\mathsf{nr}}$*, FO or DATALOG.* □

Proof sketch. The lower bounds of RPP, FRP, MBP and CPP in the absence of Q_c, for combined complexity and data complexity, respectively, obviously carry over to this setting, since when Q_c is empty (see Section 2), Q_c is in PTIME. The upper bound proofs of Theorems 4.2, 4.3, 4.4 and 4.5 for combined complexity, and Theorem 5.1, 5.2, 5.3 and 5.4 for data complexity, in the absence of Q_c, also remain intact here. Indeed, adding an extra PTIME step for checking whether $Q_c(N, D) = \emptyset$ does not increase the complexity of the algorithms given there. □

6.4. Item recommendations. As remarked in Section 2, item recommendation is a special case of package recommendation when (a) compatibility constraints Q_c are *absent*, and (b) each package consists of a single item, *i.e.,* with *a fixed size* 1. Given a database D, a query $Q \in \mathcal{L}_Q$, a rating function $f()$ and a natural number $k \geq 1$, a top-k item selection is a top-k package selection specified in terms of (Q, D, f).

When Q_c is absent and packages have size 1, one might expect that the recommendation analyses would become much simpler. Unfortunately, this is not the case.

Theorem 6.4. *For items,* RPP*,* FRP*,* MBP *and* CPP *have*

- *the same combined complexity as their counterparts in the absence of* Q_c *(Theorems 4.2, 4.3, 4.4, 4.5), and*
- *the same data complexity as their counterparts for packages with a constant bound (Corollary 6.1),*

when $\mathcal{L}_Q$ *is CQ, UCQ,* $\exists$*FO*$^+$*, DATALOG*$_{\mathsf{nr}}$*, FO or DATALOG.* □

Proof sketch. (1) Combined complexity. The upper bounds of these problems in the absence of Q_c (Theorems 4.2, 4.3, 4.4, 4.5) obviously remain intact here. The lower bound proofs for RPP and CPP given there are still valid for item recommendations, since they require only top-1 packages with a single item. For FRP and MBP, however, new lower bound proofs are required for item recommendations.

More specifically, we show that FRP(CQ) is FP$^{\mathsf{NP}}$-hard by reduction from MAX-WEIGHT SAT, and that MBP(CQ) is DP-hard by reduction from SAT-UNSAT, for item recommendations (see the proof of Theorem 4.3 for the statement of MAX-WEIGHT SAT, and the proof of Theorem 4.2 for the statement of SAT-UNSAT). For other languages $\mathcal{L}_Q$, the proofs for FRP($\mathcal{L}_Q$) and MBP($\mathcal{L}_Q$) are given along the same lines as their counterparts for Theorems 4.3 and 4.4, respectively.

(2) Data complexity. The algorithms developed for Corollary 6.1 suffice for item selections when Q is fixed. □

Table 7.1
Combined complexity ($^{(\star)}$*: items (Th.6.4),* $^{(\S)}$*: constant bound (Cor. 6.1),* $^{(\dagger)}$*:* PTIME Q_c *(Cor.6.3))*

Problems	Languages	with Q_c	Without Q_c
RPP	CQ, UCQ, $\exists$FO$^+$	Π_2^p-complete$^{(\S)}$	DP-complete$^{(\star,\dagger)}$
	DATALOG$_{\mathsf{nr}}$, FO	PSPACE-complete$^{(\S)}$	PSPACE-complete$^{(\star,\dagger)}$
	DATALOG	EXPTIME-complete$^{(\S)}$	EXPTIME-complete$^{(\star,\dagger)}$
		(Th. 4.1)	(Th. 4.2)
FRP	CQ, UCQ, $\exists$FO$^+$	FP$^{\Sigma_2^p}$-complete$^{(\S)}$	FP$^{\mathsf{NP}}$-complete$^{(\star,\dagger)}$
	DATALOG$_{\mathsf{nr}}$, FO $^{(\star,\dagger)}$	FPSPACE(poly)-complete$^{(\S)}$	FPSPACE(poly)-complete$^{(\star,\dagger)}$
	DATALOG	FEXPTIME(poly)-complete$^{(\S)}$	FEXPTIME(poly)-complete$^{(\star,\dagger)}$
		(Th. 4.3)	(Th. 4.3)
MBP	CQ, UCQ, $\exists$FO$^+$	D_2^p-complete$^{(\S)}$	DP-complete$^{(\star,\dagger)}$
	DATALOG$_{\mathsf{nr}}$, FO	PSPACE-complete$^{(\S)}$	PSPACE-complete$^{(\star,\dagger)}$
	DATALOG	EXPTIME-complete$^{(\S)}$	EXPTIME-complete$^{(\star,\dagger)}$
		(Th. 4.4)	(Th. 4.4)
CPP	CQ, UCQ, $\exists$FO$^+$	#·coNP-complete$^{(\S)}$	#·NP-complete$^{(\star,\dagger)}$
	DATALOG$_{\mathsf{nr}}$, FO	#·PSPACE-complete$^{(\S)}$	#·PSPACE-complete$^{(\star,\dagger)}$
	DATALOG	#·EXPTIME-complete$^{(\S)}$	#·EXPTIME-complete$^{(\star,\dagger)}$
		(Th. 4.5)	(Th. 4.5)

Table 7.2
Data complexity ($^{(\star)}$*: items (Th. 6.4),* $^{(\dagger)}$*:* PTIME Q_c *(Cor. 6.3))*

Problems	Poly-bounded	Constant bound
RPP	coNP-complete$^{(\dagger)}$ (Th. 5.1)	PTIME $^{(\star)}$ (Cor. 6.1)
FRP	FP$^{\mathsf{NP}}$-complete$^{(\dagger)}$ (Th. 5.2)	FP $^{(\star)}$ (Cor. 6.1)
MBP	DP-complete$^{(\dagger)}$ (Th. 5.3)	PTIME $^{(\star)}$ (Cor. 6.1)
CPP	#·P-complete$^{(\dagger)}$ (Th. 5.4)	FP $^{(\star)}$ (Cor. 6.1)

6.5. Top-1 recommendations. One may wonder whether recommending top-1 packages or items is easier than recommending top-k packages or items. However, a close inspection of the lower bound proofs of RPP, FRP and MBP reveal that these remain intact when $k = 1$ (note that k is irrelevant to CPP.) This tells us that the parameter k does not affect the complexity.

7. Conclusions.

We have studied a general model for recommendation systems, and investigated several fundamental problems in the model, from decision problems RPP, MBP to function problem FRP and counting problem CPP. We have also investigated special cases of these problems, when compatibility constraints Q_c are absent or in PTIME, when all packages are bounded by a constant B_p, and when both Q_c is absent and B_p is fixed to be 1 for item selections. We have provided a complete picture of the lower and upper bounds of these problems, *all matching*, for both their data complexity and combined complexity, when $\mathcal{L}_Q$ ranges over a variety of query languages. These results tell us where complexity of these problems arises.

The main complexity results are summarized in Tables 7.1 and 7.2 for combined complexity and data complexity, respectively, annotated with their corresponding theorems (the results for SP (Corollary 6.2) are excluded). From these results we find the following.

Query languages $\mathcal{L}_Q$. Table 7.1 tells us that query languages dominate the combined complexity of all these problems. Indeed, the presence of negation or recursion in queries makes our lives harder. In contrast, the data complexity of these problems is *independent of* query languages, and remains *unchanged* no matter whether compatibility constraints Q_c are present or not, as shown in Table 7.2.

Compatibility constraints. As we can see from Table 7.1, (1) for CQ, UCQ and $\exists\text{FO}^+$, the presence of Q_c increases the combined complexity of the analyses. (2) In contrast, for more powerful languages such as $\text{DATALOG}_{\mathsf{nr}}$, FO and DATALOG, neither Q_c nor variable sizes make any difference. Indeed, RPP, FRP, MBP and CPP have exactly the same combined complexity as their counterparts for item recommendations, in the presence or absence of Q_c. That is, the bounds for FO, $\text{DATALOG}_{\mathsf{nr}}$ and DATALOG are *robust*, *regardless of* the presence of Q_c and package sizes. (3) For data complexity, the presence of Q_c has no impact (Table 7.2). Indeed, when Q_c is fixed, it is in PTIME to check $Q_c(N, D) = \emptyset$ for all $\mathcal{L}_Q$ in which Q_c is expressed; hence Q_c can be encoded in the $\mathsf{cost}()$ function, and no longer needs to be treated separately. (4) To simplify the discussion we use $\mathcal{L}_Q$ to specify Q_c. Nonetheless, all the complexity results remain intact for any class $\mathcal{C}$ of Q_c whose satisfiability problem has the same complexity as the membership problem for $\mathcal{L}_Q$ (the satisfiability problem is to determine, given a query $Q \in \mathcal{C}$, whether there exists a database D such that $Q(D)$ is nonempty). (5) In particular, when $\mathcal{C}$ is a class of PTIME functions, the presence of Q_c has no impact on the complexity. In other words, when Q_c is a PTIME function, all these problems have the same complexity bounds as their counterparts in the absence of Q_c.

Variable sizes of packages. (1) For simple queries that have a PTIME membership problem, such as SP queries, the problems with variable package sizes have higher combined *and* data complexity than their counterparts with a fixed (constant) package size. This is in line with the claim of [34]. (2) In contrast, for any query language that subsumes CQ, variable sizes of packages have no impact on the *combined complexity* of these problems. This is consistent with the observation of [24]. (3) When it comes to *the data complexity*, however, variable (polynomially) package sizes make our lives harder: RPP, FRP, MBP and CPP in this setting have a higher data complexity than their counterparts with a fixed package size, as shown in Table 7.2.

Item recommendation. Item selections do not come with compatibility constraints Q_c and moreover, have a fixed package size 1. Observe that RPP, FRP, MBP and CPP for items have the same combined complexity as their counterparts for packages in the absence of Q_c, and they have the same data complexity as their counterparts for packages with a constant bound (see Tables 7.1 and 7.2)

The number k of packages. All the lower bounds of RPP, FRP and MBP remain intact *when* $k = 1$, *i.e.,* they carry over to top-1 package selections. That is, the number k of packages has no impact on the combined and data complexity of these problems.

The study of recommendation problems is still preliminary. First, this work aims to study a general model that subsumes previous models developed for various applications, and hence adopts generic functions $\mathsf{cost}()$, $\mathsf{val}()$ and $f()$. These need to be fine-tuned by incorporating information about users (*e.g.,* historical behaviors), collaborative filtering and specific aggregate functions. Second, to simplify the discussion we assume that selection criteria Q and compatibility constraints Q_c are expressed in the same language (albeit PTIME Q_c). It is worth studying different languages for Q and Q_c. Third, the recommendation problems are mostly intractable. An interesting topic is to identify practical and tractable cases. Finally, another issue to consider concerns group recommendations [5], to a group of users instead of a single user.

Appendix A. Proofs of Section 4.

A.1. Proof of Theorem 4.1 (RPP, combined complexity, in the presence of compatibility constraints). We prove the combined complexity bounds of $\mathsf{RPP}(\mathcal{L}_Q)$ when $\mathcal{L}_Q$ ranges over CQ, UCQ, $\exists\text{FO}^+$, $\text{DATALOG}_{\mathsf{nr}}$, FO and DATALOG.

▶ *When $\mathcal{L}_Q$ is CQ, UCQ or $\exists FO^+$.* It suffices to show that $\mathsf{RPP}(\mathcal{L}_Q)$ is Π_2^p-hard for CQ and is in Π_2^p for $\exists\text{FO}^+$.

Lower bound. To verify that $\mathsf{RPP}(\text{CQ})$ is Π_2^p-hard, we consider the *compatibility* problem. It is to determine, given Q, D, Q_c, $\mathsf{cost}()$, $\mathsf{val}()$, C and a constant B, whether there exists a nonempty $N \subseteq Q(D)$ such that $\mathsf{cost}(N) \leq C$, $\mathsf{val}(N) > B$ and $Q_c(N, D) = \emptyset$. The lower bound proof of $\mathsf{RPP}(\text{CQ})$ consists of two parts. We first show that the compatibility problem is Σ_2^p-complete for CQ queries (see Lemma A.1 below). We then verify that $\mathsf{RPP}(\text{CQ})$ is Π_2^p-hard by reduction from the complement of the compatibility problem. We first show the following lemma:

Lemma A.1. *The combined complexity of the compatibility problem is Σ_2^p-complete for CQ queries.*

Proof. We show that the compatibility problem is in Σ_2^p by giving an NP algorithm that calls an NP oracle, as follows. The algorithm first guesses a package N, where the number of tuples in N is bounded by a predefined polynomial p, and the arity of tuples in N is bounded by another predefined polynomial p_s; the algorithm then verifies whether (a) $N \subseteq Q(D)$; (b) $Q_c(N, D) = \emptyset$; and (c) $\mathsf{cost}(N) \leq C$ and $\mathsf{val}(N) > B$. When $\mathcal{L}_Q$ is CQ, UCQ or $\exists\text{FO}^+$, checking (a) and (b) requires NP and coNP, respectively. Indeed, for (a) the NP algorithm first guesses for each item $s \in N$, a CQ query Q_s from Q and a tableau from D for Q_s (see [1] for details of tableau representations of CQ queries), and then checks whether these yield N. If so, the guess is accepted and the algorithm returns "yes". For (b) the coNP algorithm simply guesses a tuple t, a CQ query Q_t from Q and a tableau D for Q_t, and then checks whether these yield the tuple t. If so, the guess is accepted and the algorithm returns "no". In addition, verifying (c) requires PTIME. From this the Σ_2^p upper bound follows.

For the lower bound, we show that the compatibility problem is Σ_2^p-hard by reduction from the $\exists^*\forall^*$3DNF problem, which is known to be Σ_2^p-complete [29]. The $\exists^*\forall^*$3DNF problem is to decide, given a sentence $\varphi = \exists X \forall Y\, \psi(X, Y)$, whether φ is true. Here $X = \{x_1, \ldots, x_m\}$, $Y = \{y_1, \ldots, y_n\}$ and ψ is a disjunction $C_1 \vee \cdots \vee C_r$, where C_i is a conjunction of three literals defined in terms of variables in $X \cup Y$.

Given an instance $\varphi = \exists X \forall Y\, \psi(X, Y)$, we define a database D, a query Q in CQ, a query Q_c in CQ for compatibility constraints, functions $\mathsf{cost}()$ and $\mathsf{val}()$, and two constants C and B, such that φ is true if and only if there exists a package $N \subseteq Q(D)$ such that $\mathsf{cost}(N) \leq C$, $\mathsf{val}(N) > B$, and $Q_c(N, D) = \emptyset$.

(1) The database D consists of four relations specified by schemas $R_{01}(X)$, $R_\vee(B, A_1, A_2)$, $R_\wedge(B, A_1, A_2)$ and $R_\neg(A, \vec{A})$. Their instances are shown in Figure A.1. More specifically, I_{01} encodes the Boolean domain, and $I_\vee$, $I_\wedge$ and $I_\neg$ encode disjunction, conjunction and negation, respectively, such that ψ can be expressed in CQ in terms of these relations.

(2) We define a CQ query Q as $Q(\vec{x}\,) = R_{01}(x_1) \wedge \cdots \wedge R_{01}(x_m)$, where $\vec{x} = (x_1, \ldots, x_m)$. That is, $Q(\vec{x}\,)$ generates all truth assignments of X variables by means of Cartesian products of R_{01}.

$$I_{01} = \begin{array}{|c|}X \\ \hline 1 \\ 0\end{array} \qquad I_{\vee} = \begin{array}{|ccc|}B & A_1 & A_2 \\ \hline 0 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1\end{array} \qquad I_{\wedge} = \begin{array}{|ccc|}B & A_1 & A_2 \\ \hline 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1\end{array} \qquad I_{\neg} = \begin{array}{|cc|}A & \bar{A} \\ \hline 0 & 1 \\ 1 & 0\end{array}$$

FIGURE A.1. *Relation instances used in the lower bound proof of Lemma A.1*

(3) We define a CQ query Q_c as follows:

$$Q_c(b) = \exists \vec{x}\ \exists \vec{y}\ \big(R_Q(\vec{x}\,) \wedge Q_Y(\vec{y}\,) \wedge Q_\psi(\vec{x}, \vec{y}, b) \wedge\ b = 0\big).$$

Here R_Q is the schema of the result of $Q(D)$, and Q_Y generates all truth assignments of Y variables by means of Cartesian products of R_{01} in the same way as $Q(\vec{x}\,)$. Query Q_ψ in CQ encodes the truth value of $\psi(X, Y)$ for given truth assignments μ_X and μ_Y, in terms of $I_\vee$, $I_\wedge$ and $I_\neg$; it returns $b = 1$ if $\psi(X, Y)$ is satisfied by μ_X and μ_Y, and $b = 0$ otherwise. Intuitively, $Q_c(b) \neq \emptyset$ if for a given set $N \subseteq Q(D)$ that encodes a truth assignment μ_X for X, there exists a truth assignment of Y that makes $\psi(\mu_X, Y)$ false.

(4) We define $\mathsf{cost}(N) = |N|$ when $N \neq \emptyset$, *i.e.,* it counts the number of items in nonempty packages N, and define $\mathsf{cost}(\emptyset) = \infty$ otherwise. In addition, we use cost budget $C = 1$, *i.e.,* any recommended package N has exactly one item. Furthermore, we let $\mathsf{val}()$ be a constant function that assigns 1 to any package and set $B = 0$.

We next verify that φ is true if and only if there exists $N \subseteq Q(D)$ such that $\mathsf{cost}(N) \leq C$, $\mathsf{val}(N) > B$, and $Q_c(N, D) = \emptyset$.

$\boxed{\Rightarrow}$ First assume that φ is true. Then there exists a truth assignment μ_X^0 for X such that for all truth assignments μ_Y for Y, ψ is true. Let N consist of the tuple representing μ_X^0. Then Q_ψ does not return $b = 0$ for μ_X^0 and hence, $Q_c(N, D)$ is empty. Obviously, $\mathsf{cost}(N) \leq C$ and $\mathsf{val}(N) > B$.

$\boxed{\Leftarrow}$ Conversely, assume that φ is false. Then for all truth assignment μ_X for X, there exists a truth assignment μ_Y for Y such that ψ is false for μ_X and μ_Y. Hence no matter how we select N, as long as N consists of a truth assignment of X, Q_ψ returns $b = 0$ and hence, $Q_c(N, D)$ is nonempty. Observe that the empty package $N = \emptyset$ cannot be recommended because $\mathsf{cost}(\emptyset) = \infty > C$.

This completes the proof of the lemma. □

We next show that RPP(CQ) is Π_2^p-hard by reduction from the complement of the compatibility problem. Given an instance Q, D, Q_c, $\mathsf{cost}()$, $\mathsf{val}()$, cost budget C and a constant B of the compatibility problem, we define a set $\mathcal{N}$ of packages, a function $\mathsf{val}'()$, and let $k = 1$. We show that there exists $N \subseteq Q(D)$ such that $\mathsf{cost}(N) \leq C$, $\mathsf{val}(N) > B$ and $Q_c(N, D) = \emptyset$ if and only if $\mathcal{N}$ is not a top-1 package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}'(), C)$.

To do this, we simply let $\mathcal{N}$ consist of a single package S, which is empty, *i.e.,* no recommendation is made. We define $\mathsf{val}'(N) = B$ if $N = S = \emptyset$, and $\mathsf{val}'(N) = \mathsf{val}(N)$ if $N \neq \emptyset$. These suffice. Indeed, first assume that $\mathcal{N}$ is a top-1 package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}'(), C)$. Then there exists no $N \subseteq Q(D)$ such that $\mathsf{cost}(N) \leq C$, $\mathsf{val}(N) > \mathsf{val}'(S) = B$, and $Q_c(N, D) = \emptyset$. Conversely, assume that $\mathcal{N}$ is not a top-1 package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}'(), C)$. Then there must exist an $N \subseteq Q(D)$ such that $\mathsf{cost}(N) \leq C$, $\mathsf{val}(N) > \mathsf{val}'(S) = B$, and $Q_c(N, D) = \emptyset$ by the definition of top-k package selections. Therefore, RPP(CQ) is Π_2^p-hard.

Upper bound. We present a Π_2^p algorithm to check whether $\mathcal{N}$ is a top-k package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$, when $\mathcal{L}_Q$ is $\exists\text{FO}^+$. The algorithm works as follows.

1. Test whether $\mathcal{N}$ is a valid package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$, in DP.
   (a) For each item s in $N_i \in \mathcal{N}$, guess a CQ query Q_s from Q and a tableau from D for Q_s. Check whether these tableaux yield $\mathcal{N}$. If so, continue; otherwise reject the guess and go back to step 1(a).
   (b) For each $N_i \in \mathcal{N}$, check whether $Q_c(N_i, D) = \emptyset$. If so, continue; otherwise return "no".
   (c) For each $N_i \in \mathcal{N}$, check whether $\mathsf{cost}(N_i) \leq C$. If so, continue; otherwise return "no".
   (d) Check whether $N_i \neq N_j$ for all $i \neq j$ and $i, j \in [1, k]$. If so, continue; otherwise return "no".
2. Test whether $\mathcal{N}$ is a top-k package selection (*i.e.,* there exists no valid package N such that (i) $N \notin \mathcal{N}$; and (ii) $\mathsf{val}(N) > \mathsf{val}(N_i)$ for some $N_i \in \mathcal{N}$) by the following Σ_2^p algorithm for the complement problem:
   (a) Guess polynomially many CQ queries from Q and for each CQ query, guess a tableau from D. These tableaux yield a package $N \subseteq Q(D)$. If $N \in \mathcal{N}$ then reject the guess and go back to step 2(a). Otherwise continue. Note that the polynomial bound on the number of queries is implied by the predefined polynomial bound on the size of packages as part of the input.
   (b) Check whether $Q_c(N, D) = \emptyset$. If so, continue; otherwise reject the guess and go back to step 2(a).
   (c) Check whether $\mathsf{cost}(N) \leq C$. If so, continue; otherwise reject the guess and go back to step 2(a).
   (d) Check whether $\mathsf{val}(N) > \mathsf{val}(N_i)$ for some $i \in [1, k]$. If so, return "no"; otherwise go back to step 2(a).

It is readily verified that step 1(a) is in NP and 1(b) is in coNP, along the same lines as in the proof of Lemma A.1. In addition, steps 1(c) and 1(d) are in PTIME. Observe that step 1 is actually in DP. Indeed, step 1 decides the yes-instances of the intersection of two languages: $\big\{\mathcal{N} = \{N_1, \ldots, N_k\} \mid$ for each $N_i \in \mathcal{N}$, $N_i \subseteq Q(D)$, $\mathsf{cost}(N_i) \leq C$ and all N_i's are pairwise distinct$\big\}$ and $\big\{\mathcal{N} = \{N_1, \ldots, N_k\} \mid$ for each $N_i \in \mathcal{N}, Q_c(N_i, D) = \emptyset\big\}$, which are in NP and coNP, respectively. Step 2 is in Π_2^p since it consists of an Σ_2^p algorithm for deciding the complement problem, *i.e.,* to find a package that has higher rating than some package in $\mathcal{N}$. Indeed, step 2(a) is an NP step that calls step 2(b), which is a coNP oracle. Furthermore, steps 2(c) and 2(d) are in PTIME. Because $\mathsf{DP} \subseteq \Pi_2^p$, the algorithm is in Π_2^p.

▶ *When $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ or FO.* We first show that $\mathsf{RPP}(\mathcal{L}_Q)$ is PSPACE-hard for DATALOG$_{\mathsf{nr}}$ or FO. We then provide a PSPACE algorithm for $\mathsf{RPP}(\mathcal{L}_Q)$ that works for both DATALOG$_{\mathsf{nr}}$ and FO.

Lower bounds. We show that RPP(DATALOG$_{\mathsf{nr}}$) and RPP(FO) are PSPACE-hard, by reduction from the membership problem for DATALOG$_{\mathsf{nr}}$ and FO, respectively. The membership problem is to determine, given a query Q in DATALOG$_{\mathsf{nr}}$ or FO, a database D and a tuple t, whether $t \in Q(D)$. It is known that this problem is PSPACE-complete for queries in DATALOG$_{\mathsf{nr}}$ [32] and FO [31]. In the following, we let $\mathcal{L}_Q$ be DATALOG$_{\mathsf{nr}}$ or FO. Given an instance (Q, D, t) of the membership problem for $\mathcal{L}_Q$, we define a query Q' in $\mathcal{L}_Q$, Q_c as the empty query, function $\mathsf{cost}(N) = |N|$ when $N \neq \emptyset$ and $\mathsf{cost}(\emptyset) = \infty$, $C = 1$, and constant function $\mathsf{val}()$ that returns 1 on each package. Furthermore, we let $N = \{t\}$ and set $k = 1$. We show that $t \in Q(D)$

if and only if $\mathcal{N} = \{N\}$ is a top-1 package selection for $(Q', D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$. It suffices to define the query

$$Q'(\vec{x}\,) \leftarrow Q(\vec{x}\,), \vec{x} = t, \quad \text{for } Q \text{ in DATALOG}_{\mathsf{nr}}\text{, or}$$
$$Q'(\vec{x}\,) \;=\; Q(\vec{x}\,) \wedge \vec{x} = t, \quad \text{for } Q \text{ in FO.}$$

Then it is readily verified that $t \in Q(D)$ if and only if $\{t\}$ is a top-1 package selection for $(Q'$, D, Q_c, $\mathsf{cost}()$, $\mathsf{val}()$, $C)$.

Upper bound. We give an NPSPACE algorithm for checking whether $\mathcal{N}$ is a top-k package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$, when Q is in DATALOG$_{\mathsf{nr}}$ or FO. It works as follows.

1. Test whether $\mathcal{N}$ is a valid package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$, in PSPACE.
   (a) Check the following: for each item s in $N_i \in \mathcal{N}$, check whether $s \in Q(D)$, in PSPACE; for each $N_i \in \mathcal{N}$, check whether $\mathsf{cost}(N_i) \leq C$; and for all $i \neq j$ and $i, j \in [1, k]$, check whether $N_i \neq N_j$, in PTIME. If all these conditions are satisfied, continue; otherwise return “no”.
   (b) For each $N_i \in \mathcal{N}$, check whether $Q_c(N_i, D) = \emptyset$. If so, continue; otherwise return “no”. This is done in PSPACE.
2. Test whether $\mathcal{N}$ is a top-k package selection, in NPSPACE.
   (a) Guess a package N consisting of polynomially many tuples of the schema R_Q of query Q.
   (b) Check the following: whether $N \subseteq Q(D)$ and $N \notin \mathcal{N}$, in PSPACE; and whether $\mathsf{cost}(N) \leq C$, in PTIME. If so, continue; otherwise reject the guess and go back to step 2(a).
   (c) Check whether $Q_c(N, D) = \emptyset$, in PSPACE. If so, continue; otherwise reject the guess and go back to step 2(a).
   (d) Check whether $\mathsf{val}(N) > \mathsf{val}(N_i)$ for some $i \in [1, k]$. If so, return “no”; otherwise go back to step 2(a).

Observe that steps 1(a), 1(b), 2(b) and 2(c) are in indeed in PSPACE since they rely on the membership problems for DATALOG$_{\mathsf{nr}}$ and FO. Including step 2(a), the overall algorithm is thus in NPSPACE when Q is in either DATALOG$_{\mathsf{nr}}$ or FO. Hence the problem is in PSPACE since NPSPACE = PSPACE [26].

▶ *When $\mathcal{L}_Q$ is DATALOG.* We show that RPP(DATALOG) is EXPTIME-complete.

Lower bound. The EXPTIME-hardness of RPP($\mathcal{L}_Q$) when $\mathcal{L}_Q$ is DATALOG is shown by reduction from the membership problem for DATALOG. The latter problem is to determine, given a DATALOG query Q, a database D and a tuple t, whether $t \in Q(D)$. It is known that this problem is EXPTIME-complete [31].

Given an instance (Q, D, t) of the membership problem for DATALOG, we define a DATALOG query Q', and Q_c as the empty query. We let $\mathsf{cost}(N) = |N|$ if $N \neq \emptyset$ and $\mathsf{cost}(\emptyset) = \infty$, $C = 1$, and let $\mathsf{val}()$ be a constant function. In addition, we set $k = 1$ and let $N = \{t\}$. Here Q' is the same as its counterpart defined in the proof for DATALOG$_{\mathsf{nr}}$ given above. It is readily verified that $t \in Q(D)$ if and only if N is a top-1 package selection for $(Q', D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$.

Upper bound. We give an EXPTIME algorithm to check whether $\mathcal{N}$ is a top-k package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$ when Q is in DATALOG:

1. Compute $Q(D)$, in EXPTIME.
2. For each $N_i \in \mathcal{N}$, check the following: (a) whether $N_i \subseteq Q(D)$, in EXPTIME, (b) whether $Q_c(N_i, D) = \emptyset$, in EXPTIME, and (c) whether $\mathsf{cost}(N_i) \leq C$, in PTIME.

For all $i \neq j$ and $i, j \in [1, k]$, check (d) whether $N_i \neq N_j$, in PTIME. If all these conditions are satisfied, continue; otherwise return "no".

3. Enumerate all subsets of $Q(D)$ consisting of polynomially many tuples. For each such set N, do the following.
   (a) Check (i) whether $N \notin \mathcal{N}$, in PTIME, (ii) whether $Q_c(N, D) = \emptyset$, in EXPTIME, and (iii) whether $\mathsf{cost}(N) \leq C$, in PTIME. If all these conditions are satisfied, continue; otherwise check the next set.
   (b) Check whether $\mathsf{val}(N) > \mathsf{val}(N_i)$ for some $i \in [1, k]$, in PTIME. If so, return "no"; otherwise check the next set.
4. Return "yes" after all the sets are inspected.

This algorithm is in EXPTIME. In particular, step 3(a) is executed exponentially many times, and still takes EXPTIME in total. Hence the problem is in EXPTIME.

This completes the proof of Theorem 4.1. Note that in all the lower bound proofs, we use $k = 1$, *i.e.,* we consider top-1 package selections. Moreover, for FO, DATALOG$_{\mathsf{nr}}$ and DATALOG, the lower bound proofs use empty compatibility constraints Q_c, *i.e.,* the lower bounds hold in the absence of Q_c. □

A.2. Proof of Theorem 4.2 (RPP, combined complexity, no compatibility constraints). Clearly, the case of $\mathsf{RPP}(\mathcal{L}_Q)$ when the compatibility constraints Q_c are absent, *i.e.,* when Q_c is the empty query, is a special case of $\mathsf{RPP}(\mathcal{L}_Q)$ in the presence of compatibility constraints. As a consequence, all upper bounds established in Theorems 4.1 carry over here. Furthermore, observe that the lower bound proofs given there do not use compatibility constraints, except for the Π_2^p-lower bound for the combined complexity of RPP(CQ). To establish the theorem we thus only need to reconsider $\mathsf{RPP}(\mathcal{L}_Q)$ when $\mathcal{L}_Q$ ranges over CQ, UCQ, or $\exists$FO$^+$.

We next show that $\mathsf{RPP}(\mathcal{L}_Q)$ is DP-hard for CQ and is in DP for $\exists$FO$^+$.

Lower bound. We show that RPP(CQ) is DP-hard by reduction from SAT-UNSAT, which is known to be DP-complete (cf. [23]). An instance of SAT-UNSAT is a pair of 3SAT instances (φ_1, φ_2), where $\varphi_1 = C_1 \wedge \cdots \wedge C_r$ is a 3SAT instance over $X = \{x_1, \ldots, x_m\}$ and φ_2 is a 3SAT instance over $Y = \{y_1, \ldots, y_n\}$. That is, φ_1 is an instance $C_1 \wedge \ldots \wedge C_r$ in which each clause C_i is a disjunction of three variables or negations thereof taken from X; similarly for φ_2 and Y. Given (φ_1, φ_2), SAT-UNSAT is to determine whether φ_1 is satisfiable and φ_2 is not satisfiable.

Given an instance (φ_1, φ_2) of SAT-UNSAT, we construct a query Q, a database D, and Q_c as the empty query. Moreover, we define $\mathsf{cost}(N) = |N|$ if $N \neq \emptyset$ and $\mathsf{cost}(\emptyset) = \infty$ and set $C = 1$. In addition, we define a rating function $\mathsf{val}()$ and a package N. Furthermore, we set $k = 1$. In other words, we consider top-1 package selections in which a package consists of one tuple. We show that $\mathcal{N} = \{N\}$ is a top-1 package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$ if and only if φ_1 is satisfiable and φ_2 is not satisfiable.

(1) The database D consists of four relations as shown in Figure A.1, specified by schemas $R_{01}(X)$, $R_\vee(B, A_1, A_2)$, $R_\wedge(B, A_1, A_2)$ and $R_\neg(A, \bar{A})$ given in the proof of Theorem 4.1. The formulas φ_1 and φ_2 can be expressed in CQ in terms of these relations.

(2) We define the CQ query Q as follows:

$$Q(b, b') \;=\; \exists \vec{x}\, \exists \vec{y}\, \big(Q_X(\vec{x}\,) \wedge Q_{\varphi_1}(\vec{x}, b) \wedge Q_Y(\vec{y}\,) \wedge Q_{\varphi_2}(\vec{y}, b')\big).$$

Here $\vec{x} = (x_1, \ldots, x_m)$ and $\vec{y} = (y_1, \ldots, y_n)$. Furthermore, the queries $Q_X(\vec{x}\,)$ and

$Q_Y(\vec{y})$ generate all truth assignments of X variables and Y variables for φ_1 and φ_2, respectively, by means of Cartesian products of R_{01}. The sub-query $Q_{\varphi_1}(\vec{x}, b)$ encodes the truth value of φ_1 for a given truth assignment μ_X such that $b = 1$ if μ_X satisfies φ_1, and $b = 0$ otherwise; similarly for $Q_{\varphi_2}(\vec{y}, b')$ and φ_2. Obviously, $Q_{\varphi_1}(\vec{x}, b)$ and $Q_{\varphi_2}(\vec{y}, b')$ can be expressed in CQ in terms of $R_\vee$, $R_\wedge$, and $R_\neg$. Observe that given D, the answer to Q in D is a subset of $\big\{(1,0),\ (1,1),\ (0,0),\ (0,1)\big\}$.

(3) It suffices to define $\mathsf{val}()$ on singleton sets. We define $\mathsf{val}(\{(1,0)\}) = 2$, $\mathsf{val}(\{(1,1)\}) = \mathsf{val}(\{(0,1)\}) = 3$ and $\mathsf{val}(\{(0,0)\}) = 1$. Furthermore, we let N consist of the single tuple $(1,0)$.

We show that $\mathcal{N} = \{N\}$ is a top-1 package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$ if and only if φ_1 is satisfiable and φ_2 is not satisfiable.

$\boxed{\Rightarrow}$ First assume that N is a top-1 package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$. Then $(1,1)$ and $(0,1)$ cannot be in $Q(D)$. Therefore, there exists a truth assignment for X making φ_1 true and moreover, there exist no assignments for Y making φ_2 true, *i.e.,* φ_1 is satisfiable and φ_2 is not satisfiable.

$\boxed{\Leftarrow}$ Conversely, assume that φ_1 is satisfiable and φ_2 is not satisfiable. Then by the definition of Q, either $Q(D) = \big\{(1,0),(0,0)\big\}$ or $Q(D) = \big\{(1,0)\big\}$. Hence $N = \big\{(1,0)\big\}$ is a top-1 package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$ by the definition of $\mathsf{val}()$ given above.

Upper bound. Consider the algorithm for $\mathsf{RPP}(\exists\text{FO}^+)$ given in the proof of Theorem 4.1 for $\exists\text{FO}^+$. Obviously, the algorithm can be applied here. Note that steps 1 and 2 are in NP and coNP, respectively, when the Q_c test is not needed (*i.e.,* without steps 1(b) and 2(b)). Thus RPP is in DP.

This completes the proof of Theorem 4.2. Observe that the lower bound proof also uses $k = 1$. □

A.3. Proof of Theorem 4.3 (FRP, combined complexity, in the presence of compatibility constraints). We start with the combined complexity of $\mathsf{FRP}(\mathcal{L}_Q)$ when $\mathcal{L}_Q$ is CQ, UCQ or $\exists\text{FO}^+$. We then consider DATALOG$_{\mathsf{nr}}$ and FO, and conclude with DATALOG.

► *When $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$.* It suffices to show that $\mathsf{FRP}(\mathcal{L}_Q)$ is $\mathsf{FP}^{\Sigma_2^p}$-hard when $\mathcal{L}_Q$ is CQ and is in $\mathsf{FP}^{\Sigma_2^p}$ when $\mathcal{L}_Q$ is $\exists\text{FO}^+$.

Lower bound. We show that $\mathsf{FRP}(\mathcal{L}_Q)$ is $\mathsf{FP}^{\Sigma_2^p}$-hard by reduction from the MAXIMUM Σ_2^p problem, which is known to be complete for the class of functions computable by a polynomial 2-alternating max-min Turing machine. This class of functions is often denoted by Σ_2^{MM} [18]. It is easily verified that a complete problem for Σ_2^{MM} is also complete for the class of functions that are computable by a $\mathsf{P}^{\Sigma_2^p}$ Turing machine, or in other words, the class of $\mathsf{FP}^{\Sigma_2^p}$ computable functions [18].

An instance of MAXIMUM Σ_2^p consists of a universally quantified formula $\varphi(X) = \forall Y \psi(X, Y)$, where $X = \{x_1, \ldots, x_m\}$, $Y = \{y_1, \ldots, y_n\}$ and ψ is a instance of 3DNF over the variables in $X \cup Y$. That is, ψ is a disjunction $C_1 \vee \cdots \vee C_r$, where each clause C_i is a conjunction of three literals over $X \cup Y$. Given φ, MAXIMUM Σ_2^p is to find the truth assignment μ_X^{last} of X that makes φ true and comes last in the lexicographical ordering on m-ary binary tuples, if it exists.

Given a MAXIMUM Σ_2^p instance φ, we construct a database D, a query Q, a query Q_c for compatibility constraints, functions $\mathsf{cost}()$ and $\mathsf{val}()$, and a cost budget

C. In particular, The database D consists of four relations as shown in Figure A.1, specified by schemas $R_{01}(X)$, $R_\vee(B, A_1, A_2)$, $R_\wedge(B, A_1, A_2)$ and $R_\neg(A, \bar{A})$ given in the proof of Theorem 4.1. Furthermore, we set $k = 1$, define $\mathsf{cost}(N) = |N|$ if $N \neq \emptyset$, $\mathsf{cost}(\emptyset) = \infty$ and set $C = 1$. That is, only packages consisting of a single tuple can be recommended. The query Q and compatibility constraint Q_c are the same as given in the proof of Lemma A.1. That is, Q returns all truth assignments of X and a package N consists of a single tuple t such that (i) t represents a truth assignment μ_X of X; and (ii) $Q_c(\{t\}, D) = \emptyset$, enforcing that μ_X makes φ true. Finally, for a tuple $t = (x_1, \ldots, x_m)$, we define $\mathsf{val}(\{t\})$ to be t, denoting the value it encodes in binary.

We next show that $\{N\}$ is a top-1 package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$, where N consists of a single tuple $t = (\vec{x}\,)$, if and only if the truth assignment μ_X encoded by $\vec{x}$ coincides with μ_X^{last}.

$\boxed{\Rightarrow}$ Assume that $t = (\vec{x}\,)$ is a top-1 package selection. Then for each $t' \in Q(D)$, $\mathsf{val}(\{t\}) \geq \mathsf{val}(\{t'\})$. As a result, the truth assignment μ_X determined by $\vec{x}$ makes φ true and has the highest rating over all such truth assignments. It suffices to observe that for any two truth assignments μ_X and μ'_X of X, μ_X comes after μ'_X in the lexicographical ordering if and only if $\mathsf{val}(t) > \mathsf{val}(t')$, where t represents μ_X and t' represents μ'_X. As a consequence, $\mu_X = \mu_X^{last}$. Note that when no top-1 package selection exists, φ is not satisfiable and hence only the empty package could be recommended. However, by the definition of the cost function, $\mathsf{val}(\emptyset) > C$, and hence no recommendation will be made.

$\boxed{\Leftarrow}$ Conversely, assume that φ is satisfiable and consider μ_X^{last}. Let t be the tuple that represents μ_X^{last}. Then by the same argument as above, $\{t\}$ will be the top-1 package by the definition of $\mathsf{val}()$. If φ is not satisfiable, then no recommendation will be returned, as argued above.

Upper bound. We show that when $\mathcal{L}_Q$ is $\exists\mathrm{FO}^+$, $\mathsf{FRP}(\mathcal{L}_Q)$ is in $\mathsf{FP}^{\Sigma_2^p}$ by providing an $\mathsf{FP}^{\Sigma_2^p}$-algorithm that on input $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, k)$, returns a top-$k$ package selection $\mathcal{N} = \{N_i \mid i \in [1, k]\}$, if it exists. That is, we develop an algorithm that runs in polynomial time with access to a Σ_2^p-oracle.

Let $\mathsf{EXISTPACK}^{\geq}(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, \mathcal{N}, N, v)$ be a procedure that returns "yes" if there exists a package $N \subseteq Q(D)$ such that $Q_c(N, D) = \emptyset$, $\mathsf{cost}(N) \leq C$, $\mathsf{val}(N) \geq v$ and N is not equal to any package already in $\mathcal{N}$. It is easily verified that this is an Σ_2^p procedure. Indeed, one simply needs to guess polynomially many tuples from D to fill in the tableaux of CQ queries obtained from Q and verify whether these produce a package N that satisfies the conditions. Since checking the conditions requires calls to an NP and a coNP oracle, the complexity of the procedure is indeed in Σ_2^p. Given this procedure, the algorithm that returns a top-k package selection $\mathcal{N} = \{N_1, \ldots, N_k\}$, if it exists, works as follows:

1. Let $B_{\max} = 2^{q(n)}$, where $q(n)$ is a polynomial that represents the length of the encoding of D and Q, taking into account that $\mathsf{cost}()$ and $\mathsf{val}()$ are PTIME functions, such that any $\mathsf{val}()$-rating of packages in $Q(D)$ lies within the interval $[0, B_{\max}]$.
2. Let $\mathcal{N} = \emptyset$ and $\ell = 1$.
3. While $\ell < k + 1$ do the following:
   (a) Perform a binary search over the interval $[0, B_{\max}]$ to find the maximal value $B \in [0, B_{\max}]$ such that there exists a valid package $N \subseteq Q(D)$ with $\mathsf{val}(N) = B$ and N is not equal to any package in $\mathcal{N}$, Clearly, B can be found in this way by making $\log(2^{q(n)}) = q(n)$ calls to the Σ_2^p oracle $\mathsf{EXISTPACK}^{\geq}(Q, D,$

$Q_c, \mathsf{cost}(), \mathsf{val}(), C, \mathcal{N}, N, v)$. Such a value will always be found, unless no k distinct packages exist. In that case, we return the empty set and terminate the algorithm. Otherwise, we continue.

(b) Given B, we know that there exists a package N such that $N \subseteq Q(D)$, $Q_c(N, D) = \emptyset$, $\mathsf{cost}(N) \leq C$, $\mathsf{val}(N) = B$ and N is distinct from any other package already in $\mathcal{N}$. It remains to find such an *optimal* package N. To do this, we proceed as follows: Let $p(n)$ be a polynomial that bounds the size of packages, *i.e.,* any package consists of at most $p(|D|)$ tuples. Let $N = \emptyset$. We will add tuples to N, one at a time, hereby guaranteeing that N can grow to an *optimal* package with $\mathsf{val}(N) = B$. Let $l = 1$.

(c) While $l < p(|D|) + 1$ do the following:

(i). We first check whether N is optimal. That is, whether $\mathsf{val}(N) = B$. If so, we add N to $\mathcal{N}$ and set $B_{\max} = B$. No further tuples need to be added to N in this case and we go to step 3 and let $\ell = \ell + 1$. That is, we extend $\mathcal{N}$ with one package (*i.e.,* N) and continue with adding a next package (if needed).

(ii). Otherwise, if $\mathsf{val}(N) < B$ then we still need to add tuples to N. We do this by gradually selecting the values for the new tuple, one attribute at a time. Let $m = \mathsf{arity}(R_Q)$ and $n = |\mathsf{adom}(Q, D)|$, where $\mathsf{arity}(R_Q)$ denotes the arity of the output schema of Q and $\mathsf{adom}(Q, D)$ is the set of constants appearing in D or Q. Denote by $\mathcal{C} = (c_{ij})$ the $m \times n$ array, where c_{ij} is the jth constant in some arbitrary ordering of $\mathsf{adom}(Q, D)$. We next show how to transform $\mathcal{C}$ into an $m \times n$ array $\mathcal{D} = (d_{ij})$ such that for each $i \in [1, m]$, there exists a unique j such that $d_{ij} = c_{ij}$, whereas for $j' \neq j$, $d_{ij'} = \sqcup$. The semantics of $\mathcal{D}$ is that we can derive a tuple $s \in Q(D)$ such that for each $i \in [1, m]$, $s[i]$ takes the unique value in the ith row of $\mathcal{D}$ different from $\sqcup$. This tuple s will be added to N. We next show how $\mathcal{D}$ is constructed from $\mathcal{C}$. Let $i = 1$, $j = 1$ and $\mathcal{D} = \mathcal{C}$.

(iii). While $i < m + 1$ and $j < n + 1$ do the following:

(A). Let $c = c_{ij}$. Consider the rating function $\mathsf{val}_{c,i,N}$ given by $\mathsf{val}_{c,i,N}(N') = B - 1$ if $N \subsetneq N'$ and N' contains a tuple $s \notin N$ with $s[i] = c$; and $\mathsf{val}_{c,i,N}(N') = \mathsf{val}(N')$ otherwise.

(B). We next call the oracle. If $\mathsf{EXISTPACK}^{\geq}(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}_{c,i,N}, C, \mathcal{N}, N, B)$ returns true, then this implies that there exists a package N', which is larger than N, $N' \subseteq Q(D)$, $\mathsf{cost}(N') \leq C$ and $\mathsf{val}_{c,i,N}(N') = B$, and N' is not equal to any package already in $\mathcal{N}$. That is, there exists an optimal N' for which we can safely assume (by the definition of $\mathsf{val}_{c,i,N}$) that $N' \setminus N$ consists of tuples that do not carry value c in their ith attribute. In other words, we can forget about any package N' such that $N' \setminus N$ carries tuples with constant c. We thus change d_{ij} to $\sqcup$ (indicating that we can ignore this value when looking for an optimal package) and set $j = j + 1$. In other words, we move to the next constant in $\mathsf{adom}(Q, D)$. Furthermore, we replace $\mathsf{val}()$ with $\mathsf{val}_{c,i,N}$, enforcing optimal extensions of N to carry values different from c.

(C). On the other hand, if $\mathsf{EXISTPACK}^{\geq}(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}_{c,i,N}, C, \mathcal{N}, N, B)$ returns false, then any optimal package N' must contain a tuple in $N' \setminus N$ that carries c in its ith attribute. In this case, we cannot disregard constant c when looking for optimal packages and thus we do not change $d_{ij} = c$ into $\sqcup$. However, we make this change for all other values $d_{ij'}$ with $j' > j$. We then set $i = i + 1$ (we move to the next attribute) and define $\mathsf{val}(N') = B - 1$ in case that N' does *not* contain a tuple in $N' \setminus N$ that carries c in its ith attribute. For all other N' we keep the original $\mathsf{val}(N')$ value. In other words, the choice of rating function will limit our search for an optimal package by only looking for packages that carry an additional tuple with a specific constant (c) in its ith attribute.

(iv). When all attributes and values are considered, we know the following: The array $\mathcal{D}$ has a unique entry that is set to a constant different from $\sqcup$ in each of its rows. Let s be the tuple obtained from these constants. Furthermore, by the construction, we know that there exists an optimal package N' such that $N \subsetneq N'$ and $N' \setminus N$ contains tuple s. We thus add s to N, reset $\mathsf{val}()$ to the original rating function and increase l by 1. That is, we extend N with one tuple and look for the next tuple to add.

4. If successful, then k packages have been added to $\mathcal{N}$ in the order of decreasing rating value. We return $\mathcal{N}$.

This algorithm runs in $\mathsf{FP}^{\Sigma_2^p}$. Indeed, the Σ_2^p-oracle $\mathsf{EXISTPACK}^{\geq}$ is called polynomially many times. We next argue for the correctness of the algorithm. First, observe that the algorithm returns a set of packages $\mathcal{N} = \{N_1, \ldots, N_k\}$ only if a top-k package selection exists. Second, the algorithm finds packages N_i in the decreasing order of their $\mathsf{val}()$-value. Indeed, in step (a), the binary search guarantees that the maximal $\mathsf{val}()$-value is selected for which there still exists a package in $Q(D)$ that differs from all previously constructed packages and that satisfies the cost budget constraint; step (c) then constructs such an optimal package in $Q(D)$ with the maximal $\mathsf{val}()$-value.

▶ *When $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$, FO or DATALOG.* We next show that $\mathsf{FRP}(\mathcal{L}_Q)$ is $\mathsf{FPSPACE}$ (poly)-complete when $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ or FO, and is $\mathsf{FEXPTIME(poly)}$-complete when $\mathcal{L}_Q$ is DATALOG.

Lower bounds. We first show that $\mathsf{FRP}(\mathcal{L}_Q)$ is $\mathsf{FPSPACE}\,(\mathsf{poly})$-hard when $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ Indeed, consider a function g in $\mathsf{FPSPACE}\,(\mathsf{poly})$. Here we mean by $\mathsf{FPSPACE}(\mathsf{poly})$ the class of functions computable by a PSPACE Turing machine that represents the output on the working tape as well [19]. Since g is in $\mathsf{FPSPACE}$ (poly), one can decide in PSPACE whether the ith bit of $g(\vec{x})$ is 1, for a given input $\vec{x}$. We show that testing whether the ith bit of $g(\vec{x})$ is set to 1 reduces to testing whether a package N_i consisting of a single tuple t_i is the top-1 package selection for $(Q_i, D_i, Q_c^i = \emptyset, \mathsf{cost}_i(), \mathsf{val}_i(), C_i)$, where Q_i is a DATALOG$_{\mathsf{nr}}$ query, D_i is a database, $\mathsf{cost}_i()$ and $\mathsf{val}_i()$ are functions and C_i is a constant. This suffices. For if it holds, then we can compute $g(\vec{x})$ by identifying each bit of its output, and hence all the functions in $\mathsf{FPSPACE}(\mathsf{poly})$ are reduced to computing top-1 package selections for DATALOG$_{\mathsf{nr}}$. From these it follows that FRP(DATALOG$_{\mathsf{nr}}$) is $\mathsf{FPSPACE}(\mathsf{poly})$-hard.

To see how to determine the ith bit of $g(\vec{x})$, we first observe that due to the PSPACE-completeness of the membership problem for DATALOG$_{\mathsf{nr}}$, there exists an instance (Q_i', D_i', t_i') of the membership problem for DATALOG$_{\mathsf{nr}}$ such that $t_i' \in Q_i'(D_i')$ if and only if the ith bit of $g(\vec{x})$ is set to 1. Next, note that the package recommendation problem for DATALOG$_{\mathsf{nr}}$ is PSPACE-complete by reduction from the membership problem as shown in the proof of Theorem 4.1. Consider the instance (D_i', Q_i', t_i'). A minor modification of the proof of Theorem 4.1 for DATALOG$_{\mathsf{nr}}$ results in a database D_i (encoding D_i'), a DATALOG$_{\mathsf{nr}}$ query Q_i (encoding Q_i' and

t'_i), the cost function $\mathsf{cost}_i(N) = |N|$ if $N \neq \emptyset$ and $\mathsf{cost}_i(\emptyset) = \infty$, $C_i = 1$, and a rating function $\mathsf{val}_i()$, such that the tuple $\{(1)\}$ is the top-1 package selection for $(Q_i, D_i, Q^i_c = \emptyset, \mathsf{cost}_i(), \mathsf{val}_i(), C_i)$ if and only if $t'_i \in Q'_i(D'_i)$ (*i.e.,* the membership problem for DATALOG$_{\mathsf{nr}}$), and tuple $\{(0)\}$ is the top-1 package selection otherwise (we enforce (0) to be always in $Q_i(D_i)$). More specifically, the construction of Q_i always adds $\{(0)\}$ to $Q_i(D_i)$ but adds the tuple $\{(1)\}$ to $Q_i(D_i)$ if and only if $t_i \in Q'_i(D'_i)$ is true. In addition, we let $\mathsf{val}_i(\{(0)\}) = 1$ and $\mathsf{val}_i(\{(1)\}) = 2$. As a consequence, $\{(1)\}$ will be recommended if it is present in $Q_i(D_i)$. Since $g(\vec{x})$ is of polynomial size, a polynomial number of instances $(Q_i, D_i, Q^i_c = \emptyset, \mathsf{cost}_i(), \mathsf{val}_i(), C_i)$ are needed to decide all the bits of $g(\vec{x})$. Assume that we need $q(|\vec{x}|)$ bits and consider the DATALOG$_{\mathsf{nr}}$ query Q that combines all Q_i such that it outputs tuples of arity $q(|\vec{x}|)$, where the ith attribute denotes the output of $Q_i(D_i)$. Furthermore, we let D consist of the union of all D_i's in which we keep the D_i's distinct by adding an identifier to each tuple, and incorporating these identifiers in the query Q. Observe that $\mathsf{cost}_i()$ does not depend on i and that C_i is always set 1. We thus let $\mathsf{cost}() = \mathsf{cost}_i()$ for some i, and set $C = 1$. Finally, we define $\mathsf{val}(\{s\}) = \sum s_i 2^i$ in binary, *i.e.,* $\mathsf{val}(\{s\}) = (s_{q(|\vec{x}|)}, \ldots, s_0)$, where s_i denotes the ith attribute value of s. In this way, tuples $s \in Q(D)$ encode values and it is now readily verified that the tuple representing $g(\vec{x})$ is the top-1 package selection for $(Q, D, Q_c = \emptyset, \mathsf{cost}(), \mathsf{val}(), C)$.

When $\mathcal{L}_Q$ is FO, we use a similar proof as in the previous case, but using FO formulas instead of queries in DATALOG$_{\mathsf{nr}}$, and by modifying the reduction to be from the membership problem for FO as given in the proof of Theorem 4.1 for FO.

When $\mathcal{L}_Q$ is DATALOG, we also use a similar proof but by using a function g in FEXPTIME(poly), *i.e.,* it is in EXPTIME to decide whether the ith bit of $g(\vec{x})$ is 1, for a given input $\vec{x}$, by using DATALOG formulas instead of queries in DATALOG$_{\mathsf{nr}}$, and by modifying the reduction to be from the membership problem for DATALOG as given in the proof of Theorem 4.1 for DATALOG.

Upper bound. We provide an FPSPACE(poly) algorithm to find a top-k package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$ when Q and Q_c are in DATALOG$_{\mathsf{nr}}$ or FO. The algorithm is a variation of the algorithm given for FRP($\exists$FO$^+$) above. Indeed, it suffices to observe that the Σ^p_2-oracle used in that algorithm can be replaced by a PSPACE oracle, when the queries and compatibility constraints involved are in DATALOG$_{\mathsf{nr}}$ or FO. As a consequence, the modified algorithm will make polynomially many calls to a PSPACE oracle and is therefore in FPSPACE(poly). Similarly, when Q and Q_c are in DATALOG, the oracle is replaced by an EXPTIME oracle to which polynomially many calls are made. Hence, when $\mathcal{L}_Q$ is DATALOG the function problem is in FEXPTIME(poly).

A.4. Proof of Theorem 4.3 (FRP, combined complexity, no compatibility constraints). When $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$, FO or DATALOG, we first show that the absence of Q_c makes no difference when combined complexity is concerned. Indeed, for the lower bounds, it suffices to observe that the reductions of the FPSPACE(poly) and FEXPTIME(poly) lower bounds given above for DATALOG$_{\mathsf{nr}}$, FO and DATALOG, respectively, do no use compatibility constraints (*i.e.,* each Q^i_c is the empty query). Furthermore, the FPSPACE(poly) and FEXPTIME(poly) algorithms given in that proof clearly remain valid after the Q_c test is removed. As a consequence, FRP($\mathcal{L}_Q$) remains FPSPACE(poly)-complete when $\mathcal{L}_Q$ is either DATALOG$_{\mathsf{nr}}$ or FO, and FEXPTIME(poly)-complete when $\mathcal{L}_Q$ is DATALOG, even in the absence of compatibility constraints.

In contrast, the absence of Q_c has an impact on the combined complexity when $\mathcal{L}_Q$

is CQ, UCQ or $\exists$FO$^+$. We show that $\mathsf{FRP}(\mathcal{L}_Q)$ is $\mathsf{FP}^{\mathsf{NP}}$-complete in the absence of Q_c, when $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$.

Lower bound. For the lower bound, it suffices to show that FRP(CQ) is $\mathsf{FP}^{\mathsf{NP}}$-hard when Q_c is absent. We verify this by reduction from MAX-WEIGHT SAT, which is known to be $\mathsf{FP}^{\mathsf{NP}}$-complete (cf. [23]). An instance of MAX-WEIGHT SAT consists of a set $\mathcal{C}$ of clauses $\{C_1, \ldots, C_r\}$ such that each clause C_i has an integer weight w_i associated with it. Furthermore, for each $i \in [1, r]$, the clause C_i is of the form $\ell_1^i \vee \ell_2^i \vee \ell_3^i$, where for each $j \in [1, 3]$, ℓ_j^i is either a variable or the negation of a variable in $X = \{x_1, \ldots, x_m\}$. Given $(\mathcal{C}, \{w_1, \ldots, w_r\})$, MAX-WEIGHT SAT is to find the truth assignment of X that satisfies a set of clauses in $\mathcal{C}$ with the largest total weight, *i.e.,* it is to find a truth assignment μ_X of X such that $\sum_{\{i | C_i(\mu_X) \text{ is true}\}} w_i$ is maximized.

Given a MAX-WEIGHT SAT instance $\mathcal{C} = \{C_1, \ldots, C_r\}$ in which each clause C_i has a weight w_i, we define a database D, a query Q in CQ, empty query Q_c, functions $\mathsf{cost}()$ and $\mathsf{val}()$, and constant C. We show that for a package $N \subseteq Q(D)$ for which $\mathsf{cost}(N) \leq C$, $\{N\}$ is a top-1 package selection if and only if N encodes the truth assignment of X that satisfies a set of clauses with the largest total weight.

(1) The database D has a single relation $R_C(\mathsf{cid}, L_1, V_1, L_2, V_2, L_3, V_3)$. Its corresponding instance I_C consists of the following set of tuples. For each $i \in [1, r]$, consider the clause $C_i = \ell_1^i \vee \ell_2^i \vee \ell_3^i$. For any possible truth assignment μ_i of variables in the literals in C_i that make C_i true, we add a tuple $(i, x_k, v_k, x_l, v_l, x_m, v_m)$, where $x_k = \ell_1^i$ in case $\ell_1^i \in X$ and $x_k = \bar{\ell}_1^i$ in case $\ell_1^i = \bar{x}_k$. We set $v_k = \mu_i(x_k)$; similarly for x_l, x_m and v_l and v_m.

(2) We take Q as the identity query on instances of R_Q.

(3) we define for each package N, $\mathsf{val}(N)$ as the sum of all the weights associated with tuples (*i.e.,* clauses) in N.

(4) For each package N, we define $\mathsf{cost}(N) = 1$ if there exists no two distinct tuples in N that have the same cid value or have different values for a variable appearing in both of them. Furthermore, for any other N, we define $\mathsf{cost}(N) = 2$. We set $C = 1$.

We next show that for a package $N \subseteq Q(D)$ for which $\mathsf{cost}(N) \leq C$, $\{N\}$ is a top-1 package selection if and only if N encodes the truth assignment of X that satisfies a set of clauses with the largest total weight. Clearly, a valid package N consists of tuples $t_1, \ldots, t_s$, at most one for each clause in φ, such that each variable in X that occurs in one of the tuples t_i has a unique value (0 or 1) in N. In other words, a package corresponds to a partial truth assignment. Clearly, the top-1 package selection will be $\{N\}$, where N is a valid package N (*i.e.,* partial truth assignment of X) that maximizes $\mathsf{val}()$. By the definition of $\mathsf{val}()$, the package that corresponds to a partial truth assignment with the largest total weight will be selected as the top-1 package. By completing the partial truth assignment in an arbitrary way, we obtain a truth assignment that maximizes the total weight of all clauses that it satisfies. Conversely, giving a truth assignment μ_X that maximizes the weights, we can easily construct a package N that consists of tuples corresponding to the clauses satisfied by μ_X. Again, $\{N\}$ will be a top-1 package selection.

Upper bound. For the upper bound, it suffices to observe that the algorithm presented for $\mathsf{FRP}(\exists\text{FO}^+)$ in the proof of Theorem 4.3 given above works for all the languages considered and moreover, the oracle used in the algorithm reduces to an NP oracle here. Indeed, the oracle guesses a package and then verifies whether (i) it is valid, (ii) has a certain rating value, and (iii) is distinct from a number of other packages. When compatibility constraints are absent, condition (i) is in NP. Conditions (ii) and (iii) are always in PTIME. As a consequence, the algorithm makes polynomially many calls to an NP oracle, from which the $\mathsf{FP}^{\mathsf{NP}}$ upper bound follows.

This completes the proof of Theorem 4.3. Note that $k = 1$ is used in all the lower bound proofs of Theorem 4.3. □

A.5. Proof of Theorem 4.4 (MBP, combined complexity, in the presence of compatibility constraints). We start with the combined complexity of $\mathsf{MBP}(\mathcal{L}_Q)$ when $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$. We then consider DATALOG$_{\mathsf{nr}}$ and FO, and conclude with DATALOG.

▶ *When $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$.* It suffices to show that $\mathsf{MBP}(\mathcal{L}_Q)$ is $\mathsf{D}_2^{\mathsf{p}}$-hard when $\mathcal{L}_Q$ is CQ and that it is in $\mathsf{D}_2^{\mathsf{p}}$ when $\mathcal{L}_Q$ is $\exists$FO$^+$.

Lower bound. We show that MBP(CQ) is $\mathsf{D}_2^{\mathsf{p}}$-hard by reduction from $\exists^*\forall^*$3DNF–$\forall^*\exists^*$3CNF, which is $\mathsf{D}_2^{\mathsf{p}}$-complete [33]. An instance of $\exists^*\forall^*$3DNF–$\forall^*\exists^*$3CNF is a pair (φ_1, φ_2) of $\exists^*\forall^*$3DNF instances as described in the proof of Lemma A.1. It is to decide whether φ_1 is true and φ_2 is false. Given (φ_1, φ_2), we define a database D, a query Q in CQ, a query Q_c in CQ for compatibility constraints, functions $\mathsf{cost}()$ and $\mathsf{val}()$ and constants C, k and B such that φ_1 is true and φ_2 is false if and only if B is the maximum bound for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, k)$. We show that this holds even when $k = 1$.

Consider (φ_1, φ_2), where $\varphi_1 = \exists X_1 \forall Y_1\, \psi_1(X_1, Y_1)$ and $\varphi_2 = \exists X_2 \forall Y_2\, \psi_2(X_2, Y_2)$. We give the reduction as follows.

(1) To start with, the database D includes four relations given by I'_{01}, $I'_{\vee}$, $I'_{\wedge}$ and $I'_{\neg}$ specified by $R'_{01}(X)$, $R'_{\vee}(C, B, A_1, A_2)$, $R'_{\wedge}(C, B, A_1, A_2)$ and $R'_{\neg}(C, A, \bar{A})$. These instances are defined in terms of I_{01} $I_{\vee}$, $I_{\wedge}$ and $I_{\neg}$ given in Figure A.1, as follows: $I'_{01} = I_{01}$, $I'_{\vee} = (\{0\} \times I_0) \cup (\{1\} \times I_{\vee})$, $I'_{\wedge} = (\{0\} \times I_0) \cup (\{1\} \times I_{\wedge})$ and $I'_{\neg} = (\{0\} \times I_0) \cup (\{1\} \times I_{\neg})$, where $I_0 = \{(0,0,0), (0,0,1), (0,1,0), (0,1,1)\}$. Intuitively, these instances encode Boolean values, disjunction, conjunction and negation, respectively, when the C-attribute is set to 1. In contrast, when the C-attribute is 0, the instance I_0 is used to generate 0 (in the B-attribute) independent of the values in the attributes A_1, A_2, A and $\bar{A}$. Furthermore, the database D includes a relation $I_c = \{(1,0), (1,1)\}$ specified by schema $R_c = (C_1, C_2)$. As will be seen shortly, I_c will be used to select truth assignments for X_1 and X_2 that make either $\forall Y_1 \psi_1(X_1, Y_1)$ true or make both $\forall Y_1 \psi_1(X_1, Y_1)$ and $\forall Y_2 \psi_2(X_2, Y_2)$ true.

(2) We define a CQ query Q as follows:

$$\begin{aligned} Q(\vec{x}_1, b_1, \vec{x}_2, b_2) \;=\; \exists \vec{y}_1, \vec{y}_2 \big(& (Q_{X_1}(\vec{x}_1) \wedge Q_{Y_1}(\vec{y}_1) \wedge Q_{\psi_1}(\vec{x}_1, \vec{y}_1, b_1) \\ & \wedge (Q_{X_2}(\vec{x}_2) \wedge Q_{Y_2}(\vec{y}_2) \;\wedge Q_{\psi_2}(\vec{x}_2, \vec{y}_2, b_2)) \wedge R_c(b_1, b_2) \big) \end{aligned}$$

where $Q_{X_1}(\vec{x}_1)$ generates all truth assignments of the X_1 variables in φ_1 by means of Cartesian products of R'_{01}; similarly $Q_{Y_1}(\vec{y}_1)$ for Y_1, $Q_{X_2}(\vec{x}_2)$ for X_2 and $Q_{Y_2}(\vec{y}_2)$ for Y_2. Query Q_{ψ_1} encodes the truth value of $\psi_1(X_1, Y_1)$ for given truth assignments μ_{X_1} and μ_{Y_1}, expressed in CQ in terms of $I'_{\vee}$, $I'_{\wedge}$ and $I'_{\neg}$ and by setting $C = 1$; it returns $b_1 = 1$ if $\psi_1(X_1, Y_1)$ is satisfied by μ_{X_1} and μ_{Y_1}, and $b_1 = 0$ otherwise. Similarly, Q_{ψ_2} encodes the truth value of $\psi_2(X_2, Y_2)$ for given truth assignments μ_{X_2} and μ_{Y_2}, where $b_2 = 1$ if $\psi_2(X_2, Y_2)$ is satisfied and $b_2 = 0$ otherwise. By leveraging R_c, each tuple

t in $Q(D)$ encodes a truth assignment μ_{X_1} (in $t[\vec{x}_1]$) and a truth assignment μ_{X_2} (in $t[\vec{x}_2]$), such that b_1 is 1 and b_2 is either 0 or 1.

(3) We define a CQ query Q_c as follows:

$$Q_c = \exists \vec{x}_1, \vec{x}_2, b_1, b_2, b'_1, b'_2, \vec{y}_1, \vec{y}_2 \big(R_Q(\vec{x}_1, b_1, \vec{x}_2, b_2) \wedge Q_{Y_1}(\vec{y}_1) \wedge Q_{Y_2}(\vec{y}_2) \\ \wedge Q'_{\psi_1}(b_1, \vec{x}_1, \vec{y}_1, b'_1) \wedge b'_1 = 0 \wedge Q'_{\psi_2}(b_2, \vec{x}_2, \vec{y}_2, b'_2) \wedge b'_2 = 0 \big)$$

Here R_Q is the schema of the result of $Q(D)$, and Q_{Y_1}, Q_{Y_2}, Q'_{ψ_1} and Q'_{ψ_2} are the same as given above except that the latter two queries carry the extra C-attribute from $I'_\vee$, $I'_\wedge$ and $I'_\neg$. For example, if $b_2 = 1$ then $Q'_{\psi_2}(b_2, \vec{x}_2, \vec{y}_2, b'_2)$ returns $b'_2 = 1$ in case that $\psi_2(X_2, Y_2)$ is satisfied for the given truth assignments for X_2 and Y_2, and it returns $b'_2 = 0$ otherwise. However, if $b_2 = 0$ then b'_2 is always 0 by the definition of $I'_\vee$, $I'_\wedge$ and $I'_\neg$. Similarly for $Q'_{\psi_1}(b_1, \vec{x}_1, \vec{y}_1, b'_1)$. By requiring $b'_1 = 0$ and $b'_2 = 0$, only falsifying truth assignments for $\psi_1(X_1, Y_2)$ and $\psi_2(X_2, Y_2)$ are considered.

Intuitively, $Q_c(N)$ is nonempty if for a given $N \subseteq Q(D)$ that encodes a truth assignment μ_{X_1} for X_1 and a truth assignment μ_{X_2} for X_2 together with b_1 and b_2, (a) in case that $b_2 = 0$, there exists a truth assignment of Y_1 that makes $\psi_1(X_1, Y_1)$ false (*i.e.,* $b'_1 = 0$); and (b) in case that $b_2 = 1$, there exists a truth assignment of Y_1 that makes $\psi_1(X_1, Y_1)$ false (*i.e.,* $b'_1 = 0$) *and* there exists a truth assignment of Y_2 that makes $\psi_2(X_2, Y_2)$ false (*i.e.,* $b'_2 = 0$). Recall that by the definition of Q, non-empty packages always have $b_1 = 1$.

(4) We define, for each package N, $\mathsf{cost}(N) = |N|$ if $N \neq \emptyset$ and $\mathsf{cost}(\emptyset) = \infty$, and set $C = 1$, *i.e.,* a valid N consists of one tuple only. Given $N = \{t\}$, we define $\mathsf{val}(N) = 1$ if the (b_1, b_2) value in t is $(1, 0)$, $\mathsf{val}(N) = 2$ if the (b_1, b_2) value in t is $(1, 1)$, and $\mathsf{val}(\emptyset) = 0$. We define bound $B = 1$.

We next verify that φ_1 is true and φ_2 is false if and only if B is the maximum bound for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, k)$, where $k = 1$.

$\boxed{\Rightarrow}$ First assume that φ_1 is true and φ_2 is false. Then there exists a truth assignment $\mu^0_{X_1}$ for X_1 such that for all truth assignment μ_{Y_1} for Y_1, ψ_1 is true, and moreover, for all truth assignments μ_{X_2} for X_2, there exists a truth assignment μ_{Y_2} for Y_2 such that ψ_2 is not satisfied by μ_{X_2} and μ_{Y_2}. Let N consist of the tuple representing $\mu^0_{X_1}$ and an arbitrary μ_{X_2}, with $(b_1, b_2) = (1, 0)$. Since $b_2 = 0$, the evaluation of $Q_c(N, D)$ returns a non-empty result if and only if there exists a truth assignment μ_{Y_1} for Y_1 such that $Q'_{\psi_1}(1, \mu^0_{X_1}, \mu_{Y_1}, 0)$ holds. Since $\mu^0_{X_1}$ makes $\forall Y_1 \psi_1(\mu_{X^0_1}, Y_1)$ true, no such μ_{Y_1} exists and thus $Q_c(N, D) = \emptyset$. Moreover, $\mathsf{val}(N) \geq B$ and $\mathsf{cost}(N) \leq C$. Hence $\mathcal{N} = \{N\}$ is a valid package selection. As a result, B is a bound for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, k)$. In addition, there exists no $B' > B$ such that B' is also a bound for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, k)$. Indeed, by the definition of $\mathsf{val}()$, the only possible N' with $\mathsf{val}(N')$ higher than B consists of some tuple in which the (b_1, b_2) value is $(1, 1)$. However, if N' is a valid package then $Q_c(N', D)$ is empty and thus since $b_2 = 1$, N' would encode a truth assignment $\mu^0_{X_2}$ for X_2 that makes $\forall Y_2 \psi_2(\mu^0_{X_2}, Y_2)$ true. This contradicts the assumption that φ_2 is false. Therefore, B is the maximum bound for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, k)$.

$\boxed{\Leftarrow}$ Conversely, assume that either φ_1 is false or φ_2 is true. We have the following cases to consider. (a) If φ_1 is false, then first assume that there exist $\mu^0_{X_1}$ for X_1 and $\mu^0_{Y_1}$ for Y_1 that make $\psi_1(\mu^0_{X_1}, \mu^0_{Y_1})$ true, *i.e.,* $b_1 = 1$. In other words, ψ_1 is not always false. However, since φ_1 is false, there must also exist a $\mu^1_{Y_1}$ for Y_1 that makes $\psi_1(\mu^0_{X_1}, \mu^1_{Y_1})$ false. Hence, for any $N = (\mu^0_{X_1}, 1, \mu_{X_2}, b_2)$, $Q_c(N, D)$ is non-empty and thus no valid package exists. Similarly, in case when ψ_1 is always false and thus $Q(D) = \emptyset$, no valid package can be selected. That is, B is not even a bound for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, k)$. (b) If φ_1 and φ_2 are both true, then there exists a truth assignment $\mu^0_{X_1}$ for X_1 such that for all truth assignment μ_{Y_1} for Y_1, ψ_1 is true, and similarly, there exists such $\mu^0_{X_2}$ for X_2. We define a package N consisting of a single tuple t_0 that encodes $\mu^0_{X_1}$ and $\mu^0_{X_2}$ with $t_0(b_1, b_2) = (1, 1)$. Then N is a valid package selection. Indeed, $\mathsf{val}(N) \geq B$ and $\mathsf{cost}(N) \leq C$, and moreover, $Q_c(N, D) = \emptyset$ since neither Q'_{ψ_1} nor Q'_{ψ_2} detects falsifying truth assignments (φ_1 and φ_2 are assumed to be true). Since $\mathsf{val}(N) = 2 > B$, B is not the maximum bound for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, k)$.

Upper bound. By the definition of maximum bound, the set of yes-instances to $\mathsf{MBP}(\exists\text{FO}^+)$ is $L_1 \cap L_2$, where

- $L_1 = \big\{(Q, D, Q_c, C, \mathsf{cost}(), \mathsf{val}(), B, k) \mid$ there exists a set $\mathcal{N} = \{N_i \mid i \in [1, k]\}$ of distinct packages such that for each $i \in [1, k]$, $N_i \subseteq Q(D)$, $\mathsf{cost}(N_i) \leq C$, $\mathsf{val}(N_i) \geq B$, N_i is of a polynomial size, and $Q_c(N_i, D) = \emptyset\big\}$; and
- $L_2 = \big\{(Q, D, Q_c, C, \mathsf{cost}(), \mathsf{val}(), B, k) \mid$ there exists *no* set $\mathcal{N}' = \{N'_i \mid i \in [1, k]\}$ of distinct packages such that for each $i \in [1, k]$, $N'_i \subseteq Q(D)$, $\mathsf{cost}(N'_i) \leq C$, $\mathsf{val}(N'_i) > B$, N'_i is of a polynomial size, and $Q_c(N'_i, D) = \emptyset\big\}$.

It suffices to show that $L_1 \in \Sigma^p_2$ and $L_2 \in \Pi^p_2$. For if it holds, then the membership in D^p_2 is immediate by the definition of D^p_2. We show that L_1 is in Σ^p_2 by giving an algorithm as follows:

1. Guess a set $\mathcal{N} = \{N_i \mid i \in [1, k]\}$ of distinct packages of polynomial sizes.
2. Check whether for each $i \in [1, k]$, $N_i \subseteq Q(D)$ (in NP), $\mathsf{cost}(N_i) \leq C$ (in PTIME), $\mathsf{val}(N_i) \geq B$ (in PTIME) and $Q_c(N_i, D) = \emptyset$ (in coNP). If so, return "yes", and otherwise reject the guess and go back to step 1.

Obviously the algorithm is in Σ^p_2, and hence so is L_1. Similarly, one can verify that L_2 is in Π^p_2.

▶ *When $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ or FO.* We next show that for DATALOG$_{\mathsf{nr}}$ and FO, MBP is PSPACE-complete.

Lower bound. We show that $\mathsf{MBP}(\mathcal{L}_Q)$ is PSPACE-hard when $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ by reduction from the membership problem for DATALOG$_{\mathsf{nr}}$ as described in the proof of Theorem 4.1 for DATALOG$_{\mathsf{nr}}$. Given an instance (Q, D, t) of the membership problem, we define a DATALOG$_{\mathsf{nr}}$ query Q', a database D, Q_c as empty query, functions $\mathsf{cost}()$, $\mathsf{val}()$, and constants $C = 1$ and $k = 1$. These are the same as their counterparts given in the proof of Theorem 4.1 for DATALOG$_{\mathsf{nr}}$. In addition, we set $B = 1$.

We show that this is indeed a reduction. To see this, first assume that $t \in Q(D)$. Then $N = \{t\}$ is a valid package. As a result, $B = 1$ is a maximum bound for $(Q', D, Q_c = \emptyset, \mathsf{cost}(), \mathsf{val}(), 1, 1)$ since $\mathsf{val}()$ assigns 1 to all packages. Conversely, if $t \notin Q(D)$ then only the empty package is recommended. Consequently, $B = 1$ is not the maximum bound for $(Q', D, Q_c = \emptyset, \mathsf{cost}(), \mathsf{val}(), 1, 1)$ since $\mathsf{cost}(\emptyset) = \infty > C$.

When $\mathcal{L}_Q$ is FO, we use a proof similar to the one given in the previous case, but using FO formulas instead of queries in DATALOG$_{\mathsf{nr}}$ and relying on the membership problem for FO as given in the proof of Theorem 4.1 for FO.

Upper bound. We show that MBP is in PSPACE for DATALOG$_{\mathsf{nr}}$ and FO. Indeed, consider the algorithm for L_1 given earlier. It is in NPSPACE for DATALOG$_{\mathsf{nr}}$ and FO. Similarly, the algorithm for L_2 is also in NPSPACE. Hence the algorithm is in NPSPACE = PSPACE, and so is MBP for FO and DATALOG$_{\mathsf{nr}}$.

▶ *When $\mathcal{L}_Q$ is DATALOG.* We show that MBP(DATALOG) is $\mathsf{EXPTIME}$-complete.

Lower bound. We show that MBP(DATALOG) is $\mathsf{EXPTIME}$-hard by reduction from the membership problem for DATALOG as in the proof of Theorem 4.1 for DATALOG. The reduction is similar to the reduction given above for $\text{DATALOG}_{\mathsf{nr}}$ and FO, but by using DATALOG queries.

Upper bound. We give an $\mathsf{EXPTIME}$ algorithm to check whether B is the maximum bound, as follows.

1. Compute $Q(D)$, in $\mathsf{EXPTIME}$.
2. Enumerate all subsets of $Q(D)$ consisting of polynomially many tuples.
3. For each $\mathcal{N}$ consisting of k such pairwise distinct subsets, and for each set N_i in $\mathcal{N}$, check: (a) whether $Q_c(N_i, D) = \emptyset$, in $\mathsf{EXPTIME}$, and (b) $\mathsf{cost}(N_i) \leq C$; and (c) whether $\mathsf{val}(N_i) \geq B$ in PTIME. If all these conditions are satisfied, continue; otherwise returns "no".
4. For each $\mathcal{N}'$ of k such pairwise distinct subsets, check conditions (a) and (b) above and in addition, condition (c'): whether $\mathsf{val}(N') > B$ for all $N' \in \mathcal{N}'$. If so, returns "no", otherwise continue to check the next $\mathcal{N}'$.
5. Return "yes" after all such sets are inspected.

Each of the steps 1–4 takes $\mathsf{EXPTIME}$. Hence MBP is in $\mathsf{EXPTIME}$ for DATALOG.

A.6. Proof of Theorem 4.4 (MBP, combined complexity, no compatibility constraints). We reconsider $\mathsf{MBP}(\mathcal{L}_Q)$ when Q_c is absent. When $\mathcal{L}_Q$ is $\text{DATALOG}_{\mathsf{nr}}$, FO or DATALOG, we show that the combined complexity of $\mathsf{MBP}(\mathcal{L}_Q)$ remain unchanged when Q_c is absent. Indeed, for the lower bounds, it suffices to observe that the proofs of the lower bounds given above for $\mathsf{MBP}(\mathcal{L}_Q)$ do not use any compatibility constraints. Furthermore, the corresponding PSPACE and $\mathsf{EXPTIME}$ upper bound proofs given earlier remain intact when Q_c is absent. Therefore, $\mathsf{MBP}(\mathcal{L}_Q)$ remains PSPACE-complete when $\mathcal{L}_Q$ is either $\text{DATALOG}_{\mathsf{nr}}$ or FO, and $\mathsf{EXPTIME}$-complete when $\mathcal{L}_Q$ is DATALOG, even in the absence of compatibility constraints.

When $\mathcal{L}_Q$ is CQ, UCQ or $\exists\text{FO}^+$, we next show that $\mathsf{MBP}(\mathcal{L}_Q)$ is DP-complete when Q_c is absent.

Lower bounds. For the lower bound, it suffices to show that MBP(CQ) is DP-hard when Q is fixed and Q_c is absent, by reduction from SAT-UNSAT (see the proof of Theorem 4.2 (CQ case) for the statement of SAT-UNSAT). Given an instance (φ_1, φ_2) defined over variables X, Y, respectively, we define D, Q, $\mathsf{cost}()$, $\mathsf{val}()$, C, B and k. We show that φ_1 is satisfiable and φ_2 is not satisfiable if and only if B is the maximum bound for $(Q, D, Q_c, C, \mathsf{cost}(), \mathsf{val}(), C, k)$ where Q_c is the empty query, *i.e.,*Q_c is absent.

(1) The database D consists of a single relation $R_C(\mathsf{cid}, L_1, V_1, L_2, V_2, L_3, V_3)$. Its corresponding instance I_C consists of the following set of tuples. For each $i \in [1, r]$, let $C_i = \ell_1^i \vee \ell_2^i \vee \ell_3^i$ be the ith clause of φ_1. Here r denotes the number of clauses in φ_1. For any possible truth assignment μ_i of variables in the literals in C_i that make C_i true, we add a tuple $(i, x_k, v_k, x_l, v_l, x_m, v_m)$, where $x_k = \ell_1^i$ in case $\ell_1^i \in X$ and $x_k = \overline{\ell}_1^i$ in case $\ell_1^i = \bar{x}_k$. We set $v_k = \mu_i(x_k)$; similarly for x_l, x_m and v_l and v_m. Similarly, we add tuples for the clauses in φ_2 but using cid values ranging from $r+1$ to $r+s$, where s denotes the number of clauses in φ_2.

(2) We define Q as the identity query over instances of R_C.

(3) We define $\mathsf{val}(N) = 1$ when N contains tuples that carry only variables in X; $\mathsf{val}(N) = 2$ if N contains tuples that carry variables in X *and* tuples that carry variables in Y; and $\mathsf{val}(N) = 0$ otherwise. We set $B = 1$ and $k = 1$.

(4) We define $\mathsf{cost}(N) = 1$ in case that (a) N contains precisely one tuple for each clause in φ_1; (b) if N additionally contains a tuple denoting a clause in φ_2, then N should also contain precisely one tuple for each clause in φ_2; and (c) all tuples in N should agree on the values of variables in X and Y. For any other N, we define $\mathsf{cost}(N) = 2$. We set $C = 1$.

We next show that φ_1 is satisfiable and φ_2 is not satisfiable if and only if B is the maximum bound for $(Q, D, Q_c, C, \mathsf{cost}(), \mathsf{val}(), C, k)$ where Q_c is the empty query.

$\boxed{\Rightarrow}$ First assume that φ_1 is satisfiable and φ_2 is not satisfiable. Let μ_X^0 be a truth assignment that makes φ_1 true. Let N consist of r tuples, one for each clause in φ_1, such that the variables in these clauses take values as given by μ_X^0. Clearly, $N \subseteq Q(D)$. Furthermore, $\mathsf{cost}(N) = 1$ and $\mathsf{val}(N) = 1$, by the definition of $\mathsf{cost}()$ and $\mathsf{val}()$. Moreover, there exists no $B' > B$ such that there exists N' with $\mathsf{cost}(N') \leq C$ and $\mathsf{val}(N') \geq B'$. Indeed, if this happens then by the definition of $\mathsf{val}()$, N' must carry both variables in X and Y, and since $\mathsf{cost}(N') \leq 1$, N' must encode a truth assignment μ_Y of Y that satisfies φ_2. But this is impossible since φ_2 is not satisfiable.

$\boxed{\Leftarrow}$ Conversely, assume that φ_1 is not satisfiable or φ_2 is satisfiable. Consider the following cases. (1) If φ_1 is not satisfiable, then by the definition of $\mathsf{cost}()$, no N can exist that carries variables in X. That is, B is not even a bound for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, k)$. (2) If φ_1 is satisfiable and φ_2 is satisfiable, we let N' be the package that encodes truth assignments μ_X^0 and μ_Y^0 that make φ_1 and φ_2 true, respectively. In other words, N' consists of $r + s$ tuples corresponding to the clauses in φ_1 and φ_2 that conform to the given truth assignments. Clearly, $\mathsf{cost}(N') = 1 \leq C$ and $\mathsf{val}(N') = 2 > B$. Hence B is not the maximum bound for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, k)$.

Upper bound. For $\exists$FO$^+$, Consider the languages L_1 and L_2 defined in the proof of Theorem 4.4 for the upper bound of $\mathsf{MBP}(\exists\text{FO}^+)$. We use the following algorithm to check membership in the language L_1:

1. Guess k sets, where each set consists of polynomially many CQ queries from Q, and for each CQ query in each set, guess a tableau from D. These tableaux yield a package $\mathcal{N} = \{N_i \mid i \in [1, k]\}$, where $N_i \subseteq Q(D)$ for all $i \in [1, k]$.
2. Check whether $\mathsf{cost}(N_i) \leq C$, $\mathsf{val}(N_i) \geq B$, and $N_i \neq N_j$ when $i \neq j$. If so, return "yes"; otherwise reject the guess and go back to step 1.

This is in NP since step 2 is in PTIME. Similarly, one can show that membership in L_2 can be decided in coNP. Hence $\mathsf{MBP}(\exists\text{FO}^+) = L_1 \cap L_2$ is in DP.

This completes the proof of Theorem 4.4. Again the lower bound proofs of Theorem 4.4 use only $k = 1$. ∎

A.7. Proof of Theorem 4.5 (CPP, combined complexity, in the presence of compatibility constraints). We start with the combined complexity of $\mathsf{CPP}(\mathcal{L}_Q)$ when $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$. We then consider DATALOG$_{\mathsf{nr}}$ and FO, and conclude with DATALOG.

▶ *When $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$.* It suffices to show that CPP(CQ) is $\#\cdot\mathsf{coNP}$-hard and $\mathsf{CPP}(\exists\text{FO}^+)$ is in $\#\cdot\mathsf{coNP}$.

Lower bound. We show that CPP(CQ) is $\#\cdot\mathsf{coNP}$-hard by parsimonious reduction from $\#\Pi_1$SAT, which is known to be $\#\cdot$coNP-complete [12]. An instance of $\#\Pi_1$SAT consists of a universally quantified Boolean formula of the form $\varphi(X,Y) = \forall X\,(C_1 \vee \cdots \vee C_r)$, where the C_i's are conjunctions of variables or negated variables taken from $X = \{x_1, \ldots, x_m\}$ and $Y = \{y_1, \ldots, y_n\}$; $\#\Pi_1$SAT is to count the number of truth assignments of Y that make φ true.

Given an instance φ of $\#\Pi_1$SAT we define a database D, Q and Q_c in CQ, $\mathsf{cost}()$ and $\mathsf{val}()$, C and B such that the number of valid packages for (Q, D, Q_c, $\mathsf{cost}()$, $\mathsf{val}()$, C, B) is equal to the number of truth assignments of Y that make φ true.

(1) The database D consists of three relations specified by schemas $R_{01}(X)$, $R_\vee(B, A_1, A_2)$ and $R_\neg(A, \bar{A})$ given in the proof of Theorem 4.1. Their corresponding instances are shown in Figure A.1.

(2) The query Q simply returns truth assignment for Y, that is,

$$Q(\vec{y}) = R_{01}(y_1) \wedge \cdots \wedge R_{01}(y_n),$$

where $\vec{y} = (y_1, \ldots, y_n)$.

(3) We consider the following CQ query Q_c:

$$Q_c(\vec{y}) = R_Q(\vec{y}) \wedge \exists \vec{x}\big(\bigwedge_{i\in[1,m]} R_{01}(x_i) \wedge \bigwedge_{i\in[1,r]} Q_{\bar{C}_i}(\vec{x},\vec{y})\big),$$

where $\vec{x} = (x_1, \ldots, x_m)$, $\vec{y} = (y_1, \ldots, y_m)$, and $Q_{\bar{C}_i}$ leverages R_{01}, $R_\vee$ and $R_\neg$ to encode the disjunctions in the negated clause $\bar{C}_i$ of C_i. The semantics of $Q_{\bar{C}_i}$ is that for a given truth assignment μ_X of X and μ_Y for Y, $Q_{\bar{C}_i}(\mu_X, \mu_Y)$ evaluates to true if $\bar{C}_i$ holds for μ_X and μ_Y; and $Q_{\bar{C}_i}(\mu_X, \mu_Y)$ returns false otherwise.

(4) We define $\mathsf{cost}(N) = |N|$ if $N \neq \emptyset$, $\mathsf{cost}(\emptyset) = \infty$, and set $C = 1$. That is, each package consists of a single tuple. Furthermore, $\mathsf{val}(N) = b$ for some constant b for all packages N. We set $B = b$.

To see that this is a parsimonious reduction, observe that $N = \{s\} \subseteq Q(D)$ if and only if s represents a truth assignment for Y in φ, as returned by Q, and in addition, $Q_c(N, D) = \emptyset$. That is, there does not exist a truth assignment μ_X of X that makes $\bar{C}_1 \wedge \cdots \wedge \bar{C}_r$ false. In other words, all truth assignments μ_X of X make at least one of the clause C_i true, and hence make φ true. Furthermore, since the condition $\mathsf{val}(N) \geq B$ does not remove any packages we have that the number of valid packages for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B)$ is equal to the number of truth assignments of Y that make φ true.

Upper bound. Consider D, Q, Q_c, $\mathsf{cost}()$, $\mathsf{val}()$, C and B as input, where Q and Q_c are in $\exists$FO$^+$. Given a package N, it is readily verified that (i) checking whether $N \subseteq Q(D)$ is in NP; (ii) testing $\mathsf{cost}(N) \leq C$ and $\mathsf{val}(N) \geq B$ is in PTIME; and (iii) checking $Q_c(N, D) = \emptyset$ is in coNP. In other words, there exist two Turing machines: an NP machine $\mathcal{M}_1$ and a coNP machine $\mathcal{M}_2$, such that $(\vec{x}, \vec{y})$ is accepted by both $\mathcal{M}_1$ and $\mathcal{M}_2$, where $\vec{x}$ is an encoding of D, Q, Q_c, C, B, $\mathsf{cost}()$ and $\mathsf{val}()$, and $\vec{y}$ is an encoding of a package N satisfying the conditions above. That is, the witness function is in DP. Furthermore, since no new values are invented by queries, the encoding $\vec{y}$ of tuples s in $Q(D)$ is bounded by $\mathsf{arity}(R_Q) \times \log|\mathsf{adom}(Q, D)|$, where $\mathsf{arity}(R_Q)$ denotes the arity of the output schema of Q, which is bounded by a predefined polynomial p_s in $|\mathcal{R}|$ and $|Q|$, and $\mathsf{adom}(Q, D)$ is the set of constants appearing in D or Q. Since packages are of size polynomial in $|D|$, we may conclude that $|\vec{y}|$ is bounded by a polynomial in $|\vec{x}|$. Putting these together, we have that $\mathsf{CPP}(\exists\mathrm{FO}^+)$ is in $\#\cdot$DP. Note however, that $\#\cdot\mathsf{DP} \subseteq \# \cdot \mathsf{P}^{\mathsf{NP}}$ simply because the $\#$ operator is monotonic in its argument. The $\#\cdot\mathsf{coNP}$ upper bound then follows from $\# \cdot \mathsf{P}^{\mathsf{NP}} = \#\cdot\mathsf{coNP}$ [15].

▶ *When $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ or FO.* We next show that $\mathsf{CPP}(\mathcal{L}_Q)$ is $\#\cdot\mathsf{PSPACE}$-complete when $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ or FO.

Lower bound. We show that $\mathsf{CPP}(\text{DATALOG}_{\mathsf{nr}})$ is $\#\cdot\mathsf{PSPACE}$-hard by parsimonious reduction from #QBF, which is $\#\cdot\mathsf{PSPACE}$-complete (implicitly given in [19]). An instance of #QBF consists of a Boolean formula of the form $\varphi = \exists X\, P_1 y_1 P_2 y_2 \cdots P_n y_n\, \psi$, where $P_1 = \forall$ and $P_i \in \{\exists, \forall\}$, for $i \in [2, n]$, and ψ is quantifier-free Boolean formula over the variables in $X = \{x_1, \ldots, x_m\}$ and $Y = \{y_1, \ldots, y_n\}$; #QBF is to count the number of truth assignments of X that make φ true.

Given an instance φ of #QBF, we construct a database D, query Q, empty compatibility constraint Q_c, functions $\mathsf{cost}()$, $\mathsf{val}()$, cost budget C and a constant B. We show that the number of valid packages for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B)$ is equal to the number of truth assignments of X that make φ true.

(1) The database D consists of a single relation I_{01} specified by schema $R_{01}(X)$, as shown in Figure A.1.

(2) We define the DATALOG$_{\mathsf{nr}}$ query Q in stages as follows. It has an IDB

$$p(\vec{x}\,) \leftarrow p_1(\vec{x}, \vec{y}).$$

Next, for each $i \in [1, n]$, $p_i(\vec{x}, \vec{y})$ is defined as follows. If P_i is $\forall$, then

$$p_i(\vec{x}, \vec{y}\,) \leftarrow p_{i+1}(\vec{x}, y_1, y_{i-1}, 1, y_{i+1}, \ldots, y_n),\ p_{i+1}(\vec{x}, y_1, y_{i-1}, 0, y_{i+1}, \ldots, y_n),$$

i.e., it checks both $y_i = 1$ and $y_i = 0$. If P_i is $\exists$, then

$$p_i(\vec{x}, \vec{y}\,) \leftarrow R_{01}(y_i),\ p_{i+1}(\vec{x}, \vec{y}\,),$$

i.e., either $y_i = 1$ or $y_i = 0$ will do. Finally, $p_{n+1}(\vec{x}, \vec{y}\,)$ is an IDB that encodes $\psi(X, Y)$, by using inequality $\neq$ to encode the negation of variables and multiple datalog rules to encode disjunction. Obviously this is a non-recursive datalog program. Observe that one could also encode the negation of variables by including an additional relation in D that encodes negation (*i.e.,* by including $R_\neg$ and $I_\neg$ as in the lower bound proof of Lemma A.1).

(3) We take $\mathsf{val}()$ to be the constant function that assigns the value 1 to every package, and define $\mathsf{cost}(N) = |N|$ in case $N \neq \emptyset$, $\mathsf{cost}(\emptyset) = \infty$, and $C = 1$.

It is readily verified that $N = \{s\} \subseteq Q(D)$ is a valid package if and only if s corresponds to a truth assignment of X that makes φ true. Furthermore, since the condition $\mathsf{val}(N) \geq B$ does not remove any packages we have that the number of valid packages for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B)$ is equal to the number of truth assignments of X that make φ true. In other words, this is indeed a parsimonious reduction.

The lower bound for $\mathsf{CPP}(\text{FO})$ is verified in the same way, but by providing a reduction from QBF by means of FO queries.

Upper bound. As implied by Theorem 4.4, the witness function for $\mathsf{MBP}(\mathcal{L}_Q)$ is in PSPACE when $\mathcal{L}_Q$ is FO or DATALOG. Hence it suffices to observe that the size of encodings of recommended packages is polynomially bounded by the size of an encoding of the input. Indeed, this readily follows from the fact that the encoding of single tuples is bounded and that each package consists of polynomially many tuples in the size of the input. Putting these together, we have that $\mathsf{CPP}(\mathcal{L}_Q)$ is in $\#\cdot\mathsf{PSPACE}$ for FO and DATALOG.

▶ *When $\mathcal{L}_Q$ is DATALOG.* We show that $\mathsf{CPP}(\text{DATALOG})$ is $\#\cdot\mathsf{EXPTIME}$-complete.

Lower bound. We show that $\mathsf{CPP}(\text{DATALOG})$ is $\#\cdot\mathsf{EXPTIME}$-hard by showing that for any function for which there exists an alternating polynomial space-bounded Turing machine $\mathcal{M} = (\text{St}, \Sigma, \delta, \imath, s_0)$ such that $h(\vec{x}) = |\{\vec{y} \mid (\vec{x}, \vec{y}) \text{ is accepted by } \mathcal{M}\}|$ and $|\vec{y}| \leq |\vec{x}|^k$ for some k, there exist Q, D, empty compatibility constraint Q_c, functions $\mathsf{cost}()$, $\mathsf{val}()$, and constants C and B such that the number of valid packages for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B)$ is equal to $h(\vec{x})$. In particular, we set $\mathsf{cost}(N) = |N|$ if $N \neq \emptyset$, $\mathsf{cost}(\emptyset) = \infty$, and $C = 1$. In addition, $\mathsf{val}()$ assigns the same value b to all packages. We let $B = b$. Note that packages consist of a single tuple only and since Q_c is the empty query, a tuple makes a valid package if and only if it belongs to $Q(D)$.

Recall that $\mathsf{EXPTIME}$ coincides with languages accepted by alternating polynomial space-bounded Turing machines. An alternating Turing machine (ATM) is of the form $\mathcal{M} = (\text{St}, \Sigma, \delta, \imath, s_0)$, where St is a set of states with initial state s_0; Σ is a finite tape alphabet; transition function $\delta : \text{St} \times \Sigma \to 2^{\text{St}\times\Sigma\times\{L,R\}}$ and $\imath : \text{St} \to \{\wedge, \vee, \mathsf{acc}, \mathsf{rej}\}$. Here L shifts the head to the left and R shifts it to the right. If $\mathcal{M}$ is in a configuration with state s and $\imath(s) = \mathsf{acc}$, then that configuration is accepting; if it is in a state s with $\imath(s) = \mathsf{rej}$ then the configuration is rejecting. A configuration with s such that $\imath(s) = \wedge$ is accepting if all configurations reachable in one step are accepting; and it is rejecting otherwise. A configuration with s such that $\imath(s) = \vee$ is accepting if one of the configurations reachable in one step is accepting; and it is rejecting otherwise. An ATM $\mathcal{M}$ accepts a string $\vec{x}$ if the initial configuration with state s_0 and head positioned to the left of the input string $\vec{x}$ is accepting.

Let h be a function for which there exists an alternating polynomial space-bounded Turing machine $\mathcal{M} = (\text{St}, \Sigma, \delta, \imath, s_0)$ such that $h(\vec{x}) = |\{\vec{y} \mid (\vec{x}, \vec{y}) \text{ is accepted by } \mathcal{M}\}|$ and $|\vec{y}| \leq |\vec{x}|^k$ for some k. Let q be a polynomial such that $\mathcal{M}$ on an input of length n uses at most $g(n) = n^k + q(n)$ cells of its tape for input $\vec{x}$ and computation; the first n^k cells are reserved for $\vec{y}$ (possible padded with blanks $\sqcup$ to fill all n^k cells). Let $\vec{x} = (x_1, \ldots, x_n)$ be an input string.

Given $\mathcal{M}$ and $\vec{x}$, we define a database D, a DATALOG query Q, and functions and constants as specified above. The database D consists of a unary relation R_Σ that encodes the alphabet Σ. The query Q is defined as follows:

- If $\imath(s) = \vee$, then for each $s' \in \text{St}$, for each $a, a' \in \Sigma$ with $\delta(s, a) = (s', a', \mu)$, for some $\mu \in \{L, R\}$, and for each $i \in [0, g(|\vec{x}|)]$, we add the following rule:
$$\Pi_{s,i}(z_1, \ldots, z_{i-1}, a, z_{i+1}, \ldots, z_{g(|\vec{x}|)}) \leftarrow \Pi_{s',i+\ell}(z_1, \ldots, z_{i-1}, a', z_{i+1}, \ldots, z_{g(|\vec{x}|)}),$$
where $\ell = -1$ if $\mu = L$, and $\ell = 1$ otherwise.
- If $\imath(s) = \wedge$, then for each $a \in \Sigma$, we construct the set $Q_{s,a} = \{(s_1, a_1), \ldots, (s_{k(a)}, a_{k(a)})\}$ consisting of all pairs (s_j, a_j) such that $\delta(s, a) = (s_j, a_j, \mu_j)$ for some $\mu_j \in \{L, R\}$. As before, we set $\ell_j = -1$ if $\mu_j = L$ and $\ell_j = 1$ otherwise. For each $i \in [0, g(|\vec{x}|)]$, we add the following rule:
$$\Pi_{s,i}(z_1, \ldots, z_{i-1}, a, z_{i+1}, \ldots, z_{g(|\vec{x}|)}) \leftarrow \bigwedge_{j=1}^{k(a)} \Pi_{s_j,i+\ell_j}(z_1, \ldots, z_{i-1}, a_j, z_{i+1}, \ldots, z_{g(|\vec{x}|)}).$$
- If $\imath(s) = \mathsf{acc}$ is an accepting state then for each $i \in [0, g(|\vec{x}|)]$, we add the following rule:
$$\Pi_{s,i}(z_1, \ldots, z_{g(|\vec{x}|)}) \leftarrow \bigwedge_{j=1}^{g(|\vec{x}|)} R_\Sigma(z_i),$$

where R_Σ denotes the unary instance consisting of all alphabet symbols.

- If $\imath(s) = \mathsf{rej}$ is a rejecting state then for each $i \in [0, g(|\vec{x}|)]$, we add the following rule:

$$\Pi_{s,i}(z_1, \ldots, z_{g(|\vec{x}|)}) \leftarrow \emptyset.$$

- If $s = s_0$ then we add

$$Q(y_1, \ldots, y_{n^k}) \leftarrow \Pi_{s_0,0}(y_1, \ldots, y_{n^k}, x_1, \ldots, x_n, \sqcup, \ldots, \sqcup).$$

Clearly, $\vec{y} \in Q(D)$ if and only if $(\vec{x}, \vec{y})$ is accepted by $\mathcal{M}$. Here, $\vec{y} = (y_1, \ldots, y_{n^k})$ is the tuple encoding $\vec{y}$. Furthermore, $|Q(D)| = |\{\vec{y} \mid (\vec{x}, \vec{y}) \text{ is accepted by } \mathcal{M}\}| = h(\vec{x})$.

Upper bound. As shown by Theorem 4.4, the witness function for MBP(DATALOG) is in EXPTIME. Hence it suffices to observe that the size of encodings of recommended packages is polynomially bounded by the size of encoding of the input. Indeed, this readily follows from the fact that the encoding of single tuples is bounded, and packages consist of polynomially many tuples in the size of the input. Therefore, CPP(DATALOG) is in #·EXPTIME.

A.8. Proof of Theorem 4.5 (CPP, combined complexity, no compatibility constraints). When $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$, FO or DATALOG, we show that the absence of Q_c makes no difference when the combined complexity is concerned. In contrast, the absence of Q_c does have an effect when $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$.

▶ *When $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$.* it suffices to show that CPP(CQ) is #·NP-hard and that CPP($\exists$FO$^+$) is in #·NP.

Lower bound. We show that CPP(CQ) is #·NP-hard by parsimonious reduction from #Σ_1SAT, which is known to be #·NP-complete [12]. An instance of #Σ_1SAT consists of an existentially quantified Boolean formula of the form $\varphi(X, Y) = \exists X\,(C_1 \wedge \cdots \wedge C_r)$, where C_i are disjunctions of variables or negated variables taken from $X = \{x_1, \ldots, x_m\}$ and $Y = \{y_1, \ldots, y_n\}$; #Σ_1SAT is to count the number of truth assignments of Y that make φ true.

Given an instance φ of #Σ_1SAT, we define a database D, a CQ query Q, empty compatibility constraints Q_c, functions $\mathsf{cost}()$, $\mathsf{val}()$, and constants C and B. We show that the number of valid packages for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B)$ is equal to the number of truth assignments of Y that make φ true. In particular, we let $\mathsf{cost}(N) = |N|$ if $N \neq \emptyset$, $\mathsf{cost}(\emptyset) = \infty$ and we set $C = 1$. In addition, $\mathsf{val}()$ is a constant function assigning a value b to all packages. We let $B = b$. Note that valid packages consist of a single tuple only.

(1) The database consists of four relations specified by schemas $R_{01}(X)$, $R_\vee(B, A_1, A_2)$, $R_\wedge(B, A_1, A_2)$, and $R_\neg(A, \bar{A})$ given in the proof of Theorem 4.1. The corresponding instances are shown in Figure A.1.

(2) The query Q is then given by:

$$Q(\vec{y}) = \exists \vec{x} \Big(\bigwedge_{i \in [1,n]} R_{01}(y_i) \wedge \bigwedge_{i \in [1,m]} R_{01}(x_i) \wedge \bigwedge_{i \in [1,r]} Q_i(\vec{x}, \vec{y}) \Big),$$

where $\vec{x} = (x_1, \ldots, x_m)$, $\vec{y} = (y_1, \ldots, y_n)$, and Q_i leverages R_{01}, $R_\vee$ and $R_\neg$ to encode the disjunctions in the clause C_i. The semantics of Q_i is that for a given truth assignment μ_X of X and μ_Y for Y, $Q_i(\mu_X, \mu_Y)$ evaluates to true if C_i holds for μ_X and μ_Y; and $Q_i(\mu_X, \mu_Y)$ returns false otherwise.

To see that this is a parsimonious reduction, observe that a package N can be recommended if and only if N consists of a single tuple $s \in Q(D)$. Note that $s \in$

$Q(D)$ if and only if s represents a truth assignment for Y in φ that makes φ true. Furthermore, since the condition $\mathsf{val}(\{s\}) \geq B$ does not remove any tuples, we have that the number of valid packages for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B)$ is equal to the number of truth assignments of Y that make φ true.

Upper bound. It is readily verified that $\mathsf{CPP}(\exists\mathrm{FO}^+)$ is in $\#\cdot\mathsf{NP}$, simply because verifying whether a given package is valid is in NP in the absence of compatibility constraints.

▶ *When $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$, FO or DATALOG.* It suffices to observe that the proofs of the $\#\cdot\mathsf{PSPACE}$ and $\#\cdot\mathsf{EXPTIME}$ lower bounds of $\mathsf{CPP}(\mathcal{L}_Q)$ for DATALOG$_{\mathsf{nr}}$, FO and DATALOG given earlier do no use compatibility constraints. Together with the upper bounds given there, we conclude that $\mathsf{CPP}(\mathcal{L}_Q)$ is $\#\cdot\mathsf{PSPACE}$-complete when $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ or FO, and it is $\#\cdot\mathsf{EXPTIME}$-complete when $\mathcal{L}_Q$ is DATALOG.

This completes the proof of Theorem 4.5. □

Appendix B. Proofs of Section 5.

B.1. Proof of Theorem 5.1 (RPP, data complexity). It suffices to show that the data complexity of $\mathsf{RPP}(\mathcal{L}_Q)$ is already coNP-hard when $\mathcal{L}_Q$ is CQ and Q_c is absent, and in coNP when $\mathcal{L}_Q$ is DATALOG or FO and Q_c may be present.

Lower bound. The coNP-hardness of $\mathsf{RPP}(\mathrm{CQ})$ is shown as follows. First we prove that the data complexity of the compatibility problem is NP-complete for CQ queries. We then verify that $\mathsf{RPP}(\mathrm{CQ})$ is coNP-hard by reduction from the complement of the compatibility problem when Q_c is the empty query, as defined in the proof of the combined complexity of $\mathsf{RPP}(\mathrm{CQ})$ in Theorem 4.1.

Lemma B.1. *The data complexity of the compatibility problem is* NP*-complete for CQ queries.*

Proof. The NP upper bound for the compatibility problem follows from the following NP algorithm for the compatibility problem: (i) simply guess a package; and (ii) test whether it satisfies the condition. We verify the NP lower bound by reduction from 3SAT, which is known to be NP-complete (cf. [23]). An instance φ of 3SAT is a formula $C_1 \wedge \cdots \wedge C_r$ in which each clause C_i is a disjunction of three variables or negations thereof taken from $X = \{x_1, \ldots, x_n\}$. Given φ, 3SAT is to decide whether φ is satisfiable, *i.e.,* whether there exists a truth assignment for variables in X that satisfies φ.

Given an instance φ of 3SAT above, we define the same database D, identity query Q, empty query Q_c, and function $\mathsf{cost}()$ as their counterparts given in the lower bound proof of Theorem 4.3 for $\mathsf{FRP}(\mathrm{CQ})$, where Q_c is absent. That is, each clause C_i is encoded by tuples in D such that these encode all truth assignments for variables in C_i that make C_i true, and moreover, for each package N such that $\mathsf{cost}(N) \leq 1$, N encodes a valid truth assignment for (part of) the variables in X that make some clauses in $\{C_1, \ldots, C_r\}$ true. Furthermore, we set $C = 1$ and $k = 1$. Finally, We define for each package N, $\mathsf{val}(N) = |N|$ and set $B = r - 1$. That is, any package must consist of at least r tuples.

We next verify that φ is true if and only if there exists an $N \subseteq Q(D)$ such that $\mathsf{cost}(N) \leq C$ and $\mathsf{val}(N) > B$.

$\boxed{\Rightarrow}$ First assume that φ is satisfiable. Then there exists a truth assignment μ_X^0 for X that satisfies φ, *i.e.,* every clause C_j of φ is true by μ_X^0. Let N consist of r tuples from D, one for each clause, in which the values for the variables correspond to μ_X^0. Then $\mathsf{val}(N) = r > B$ and $\mathsf{cost}(N) = 1 \leq C$.

$\boxed{\Leftarrow}$ Conversely, assume that φ is not satisfiable. Suppose by contradiction that there exists an $N \subseteq Q(D)$ with $\mathsf{cost}(N) \leq C$ and $\mathsf{val}(N) > B$. Then N consists of r tuples and since $\mathsf{cost}(N) \leq C$ one can construct a truth assignment μ_N for X that makes all clauses in φ true. A contradiction. This completes the proof of the Lemma B.1. □

One can verify that $\mathsf{RPP}(\mathrm{CQ})$ is coNP-hard by using the same argument given in the proof of Theorem 4.1 for the combined complexity of $\mathsf{RPP}(\mathrm{CQ})$, by using Lemma B.1 and letting $k = 1$, where Q_c is the empty query.

Upper bound. We show that $\mathsf{RPP}(\mathcal{L}_Q)$ is in coNP (data complexity) for all the query languages considered when Q_c may be present. Observe that the algorithm for $\mathsf{RPP}(\mathrm{FO})$ (combined complexity, proof of Theorem 4.1) works correctly for all those query languages. Indeed, it suffices to observe that when data complexity is concerned, steps 1(a) and 1(b) are in PTIME, and similarly, steps 2(b) and 2(c) are in PTIME. This follows from the fact that the data complexity of the membership problem for $\mathcal{L}_Q$ is in PTIME, even when $\mathcal{L}_Q$ is FO or DATALOG. Since step 2 involves guessing a package and deciding the existence of package with higher rating than some package in $\mathcal{N}$ (*i.e.,* the complement problem), the algorithm is thus in coNP overall.

This completes the proof of Theorem 5.1. Note that in the lower bound proof, only $k = 1$ is used. □

B.2. Proof of Theorem 5.2 (FRP, data complexity). We show that $\mathsf{FRP}(\mathcal{L}_Q)$ is $\mathsf{FP}^{\mathsf{NP}}$-complete for data complexity when $\mathcal{L}_Q$ ranges over the languages CQ, UCQ, $\exists\mathrm{FO}^+$, $\mathrm{DATALOG}_{\mathsf{nr}}$, FO and DATALOG, in the presence or absence of compatibility constraints.

For the lower bound, it suffices to show that $\mathsf{FRP}(\mathrm{CQ})$ is $\mathsf{FP}^{\mathsf{NP}}$-hard when Q_c is absent. Observe that in the proof of Theorem 4.3, $\mathsf{FRP}(\mathrm{CQ})$ is $\mathsf{FP}^{\mathsf{NP}}$-hard when Q_c is absent. Furthermore, the reduction given there use a fixed query Q in CQ (*i.e.,* a fixed identity query). As a consequence, the data complexity of $\mathsf{FRP}(\mathrm{CQ})$ is $\mathsf{FP}^{\mathsf{NP}}$-hard when Q_c is absent.

For the upper bound, it suffices to observe that the algorithm presented for $\mathsf{FRP}(\exists\mathrm{FO}^+)$ in the proof of Theorem 4.3 works for all the languages considered and moreover, the oracle used in the algorithm reduces to an NP oracle. Indeed, the oracle guesses a package and then verifies whether (i) it is valid, (ii) it has a certain rating value, and (iii) it is distinct from the other packages. When data complexity is concerned, condition (i) is in PTIME for all considered languages. Conditions (ii) and (iii) are always in PTIME. As a consequence, the algorithm makes polynomially many calls to an NP oracle, from which the $\mathsf{FP}^{\mathsf{NP}}$ upper bound follows.

This completes the proof of Theorem 5.2. Again only $k = 1$ is needed in the lower bound proof. □

B.3. Proof of Theorem 5.3 (MBP, data complexity). We next show the data complexity of $\mathsf{MBP}(\mathcal{L}_Q)$ in the presence or absence of Q_c. It suffices to show that $\mathsf{MBP}(\mathrm{CQ})$ is DP-hard when Q is a fixed CQ query and Q_c is absent, and $\mathsf{MBP}(\mathcal{L}_Q)$ is in DP for fixed DATALOG and FO queries Q and Q_c.

For the lower bound, it is already shown in the proof of the Theorem 4.4 that $\mathsf{MBP}(\mathrm{CQ})$ is DP-hard in the absence of Q_c. Furthermore, the reduction in that proof uses a fixed query Q in CQ (*i.e.,* a fixed identity query) and empty compatibility constraint. Thus the data complexity of MBP(CQ) is also DP-hard when Q_c is absent.

For the upper bound, we show that MBP is in DP when Q and Q_c are fixed DATALOG or FO queries and Q_x may be present. Consider corresponding languages L_1 and L_2 defined in the upper bound proof of Theorem 4.4 for $\mathsf{MBP}(\exists\mathrm{FO}^+)$. When Q and Q_c are fixed, L_1 is in NP and L_2 is in coNP, for $\mathcal{L}_Q$ ranging over all the languages considered. As the set of yes-instances of MBP is $L_1 \cap L_2$, MBP is in DP.

This completes the proof of Theorem 5.3. The lower bound proof also uses $k = 1$ only. □

B.4. The proof of Theorem 5.4 (CPP, data complexity). We show that the data complexity of $\mathsf{CPP}(\mathcal{L}_Q)$ is $\#\cdot\mathsf{P}$-complete for all the query languages considered whenever Q_c is absent or present. It suffices to show that for fixed Q and Q_c, CPP(CQ) is $\#\cdot\mathsf{P}$-hard when Q_c is absent, and CPP(FO) and CPP(DATALOG) are in $\#\cdot\mathsf{P}$ when Q_c may be present.

Lower bound. We show $\#\cdot\mathsf{P}$-hardness by parsimonious reduction from #SAT, which is known to be $\#\cdot\mathsf{P}$-complete (recall that $\#\mathsf{P} = \#\cdot\mathsf{P}$). An instance of #SAT is an instance $\varphi(X) = C_1 \wedge \cdots \wedge C_r$ of 3CNF over $X = \{x_1, \ldots, x_m\}$. It is to count the number of truth assignments of X that make φ true.

Given φ, we define a database D, an identity query Q, empty compatibility constraints Q_c, functions $\mathsf{cost}()$, $\mathsf{val}()$ and constant $C = 1$. These are the same as their counterparts given in the proof of Lemma B.1. Furthermore, we set $B = r$. From that proof, we know that for a package $N \subseteq Q(D)$, $\mathsf{cost}(N) \leq C$ and $\mathsf{val}(N) \geq B$ if and only if N encodes a truth assignment for X variables that make φ true.

Hence, the number of valid packages for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B)$ is equal to the number of truth assignments of X that satisfy φ.

Upper bound. Given D, Q, Q_c, $\mathsf{cost}()$, $\mathsf{val}()$, C and B and a package N, verifying whether N is a valid package is in PTIME. Furthermore, since N is polynomially bounded by $|D|$, N consists of values from the active domains of D and Q, and the arity of the tuples in N is polynomially bounded by $|Q|$ and $|\mathcal{R}|$, one can easily verify that the size of encoding of packages is polynomially bounded by the size of encoding of the input. In other words, $\mathsf{CPP}(\mathcal{L}_Q)$ is in $\#\cdot\mathsf{P}$ for all languages considered.

This completes proof of Theorem 5.4. □

Appendix C. Proofs of Section 6.

C.1. Proof of Corollary 6.1 (Packages with a fixed bound). For combined complexity, we observe that the lower bound proofs of RPP, FRP, MBP and CPP given in Theorems 4.1, 4.3, 4.4 and 4.5, respectively, use only top-1 packages with one item. They thus carry over to the setting when the size of packages is fixed. For the upper bounds, the algorithms given there obviously remain intact in the special case for packages with a constant bound B_p.

For the data complexity, it suffices to show that, for fixed DATALOG and FO queries Q and Q_c, and packages with a constant bound B_p, RPP, FRP, MBP and CPP are in PTIME, FP, PTIME and FP, respectively. Let $\mathcal{L}_Q$ be either DATALOG or FO.

(a) $\mathsf{RPP}(\mathcal{L}_Q)$. Consider the algorithm given in the proof of Theorem 4.1 for RPP (DATALOG). We revise the algorithm such that in step 3, it only enumerates all

subsets of $Q(D)$ consisting of B_q tuples at most, and do steps 3(a) and 3(b) of the algorithm for each of these subsets. Clearly, the revised algorithm works here, even for FO. We next show that it is a PTIME algorithm. Note that steps 1 and 2 are in PTIME for fixed queries Q and Q_c in DATALOG or FO. Furthermore, there are polynomial many subsets of $Q(D)$ consisting of B_q tuples. So step 3 is also in PTIME. Thus the algorithm is in PTIME.

(b) $\mathsf{FRP}(\mathcal{L}_Q)$. Observe that the algorithm given in the proof of Theorem 4.3 for FRP ($\exists$FO$^+$) works here, when $\mathcal{L}_Q$ is DATALOG or FO. Moreover, it is easy to see that the oracle used in the algorithm reduces to a PTIME oracle. Indeed, since we only consider packages with constant bound B_p, the oracle only needs to enumerate polynomial many guesses of subsets of $Q(D)$. Moreover, when data complexity is considered, for each guessed package N, it is PTIME to check whether (i) it is valid, (ii) it has a certain rating value, and (iii) it is distinct from the other packages. Thus the algorithm is in FP. Hence the problem is in FP.

(c) $\mathsf{MBP}(\mathcal{L}_Q)$. To verify that MBP(FO) and MBP(DATALOG) are both in PTIME, we use the algorithm given in the proof of Theorem 4.4 for MBP(DATALOG) (combined complexity), except that in step 2, the algorithm enumerates all subsets of $Q(D)$ consisting of B_q tuples at most. Then there are only polynomially many such subsets. Clearly, the algorithm works here, and moreover, it is easy to see that the algorithm is in PTIME for data complexity.

(d) $\mathsf{CPP}(\mathcal{L}_Q)$. We give an FP algorithm. Given D, Q, Q_c, $\mathsf{cost}()$, $\mathsf{val}()$, C, B, and when $\mathcal{L}_Q$ is FO or DATALOG, it is to count the number of packages that are valid for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B)$. It works as follows:

1. Denote by n the number of packages that are valid for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B)$. Initially, let $n = 0$.
2. Compute $Q(D)$.
3. Enumerate all subsets of $Q(D)$ consisting of B_q tuples at most.
4. For each such subset N, check whether N is valid for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B)$. If so, let $n = n + 1$; otherwise, continue.
5. Return n after all the subsets of $Q(D)$ are inspected.

Step 1 is in PTIME for fixed Q. Furthermore, there are polynomially many subsets enumerated in step 2, and moreover, it can be readily verified that step 4 is also in PTIME for fixed Q and Q_c. Thus the algorithm is in FP.

This completes the proof of Corollary 6.1. □

C.2. The proof of Corollary 6.2 (SP queries). We first show the complexity results of RPP, FRP, MBP and CPP, respective, for packages of variable sizes, and then for packages with a constant bound.

For packages of variable sizes, it suffices to show that problems $\mathsf{RPP}(\mathcal{L}_Q)$, $\mathsf{FRP}(\mathcal{L}_Q)$, $\mathsf{MBP}(\mathcal{L}_Q)$ and $\mathsf{CPP}(\mathcal{L}_Q)$ are coNP-hard, $\mathsf{FP}^{\mathsf{NP}}$-hard, DP-hard and $\# \cdot \mathsf{P}$-hard for fixed SP queries, respectively, when Q_c is absent, and are in coNP, $\mathsf{FP}^{\mathsf{NP}}$, DP and $\# \cdot \mathsf{P}$, respectively, for varied SP queries, when Q_c is present. Observe that the lower bounds of $\mathsf{RPP}(\mathcal{L}_Q)$, $\mathsf{FRP}(\mathcal{L}_Q)$, $\mathsf{MBP}(\mathcal{L}_Q)$ and $\mathsf{CPP}(\mathcal{L}_Q)$ for data complexity, in the absence of Q_c, given in Theorem 5.1, 5.2, 5.3 and 5.4, respectively, are established by taking Q as a identity query, which is in SP. As a result, these lower bounds hold here. For the upper bound, obviously, the algorithms for RPP(FO), FRP($\exists$FO$^+$), MBP(FO), given in Theorem 5.1, 5.2 and 5.3, respectively, can carry over here. Furthermore, since the combined complexity of membership problem of SP queries is in PTIME, one can readily verify that these algorithms are in coNP, $\mathsf{FP}^{\mathsf{NP}}$ and DP, respectively. For $\mathsf{CPP}(\mathcal{L}_Q)$, it is easy to see that it is in PTIME to check whether a given set N is valid for (Q, D, Q_c, cost(), val(), C, B). Thus the problem is in #·P.

For packages of a constant size, it suffices to show that $\mathsf{RPP}(\mathcal{L}_Q)$, $\mathsf{FRP}(\mathcal{L}_Q)$, $\mathsf{MBP}(\mathcal{L}_Q)$ and $\mathsf{CPP}(\mathcal{L}_Q)$ are in PTIME, FP, PTIME and FP, respectively, for varied queries Q and Q_c in SP. Since the combined complexity of membership problem of SP queries is in PTIME, obviously, the algorithms for fixed Q and Q_c given in Corollary 6.1 can carry over here.

This completes the proof of Corollary 6.2. □

C.3. The proof of Corollary 6.3 (PTIME compatibility constraints). The lower bounds of RPP, FRP, MBP and CPP in the absence of Q_c, given in Theorems 4.2, 4.3, 4.4 and 4.5 for combined complexity, respectively, and in Theorem 5.1, 5.2, 5.3 and 5.4 for data complexity, respectively, obviously carry over to this setting, since when Q_c is empty (see Section 2), Q_c is in PTIME. The upper bound proofs given there for combined complexity (Theorems 4.2, 4.3, 4.4 and 4.5) and for data complexity (Theorem 5.1, 5.2, 5.3 and 5.4), in the absence of Q_c, also remain intact here. Indeed, adding an extra PTIME step for checking $Q_c(N, D) = \emptyset$ does not increase the complexity of the algorithms given there.

This completes the proof of Corollary 6.3. □

C.4. The proof of Theorem 6.4 (Item recommendations). We first verify the combined complexity results of RPP, FRP, MBP and CPP for items, and then show their data complexity.

Combined complexity. For the combined complexity, we observe that the upper bounds of RPP, FRP, MBP and CPP, in absence of compatibility constraints, given in Theorems 4.2, 4.3, 4.4 and 4.5, respectively, obviously remain intact here. Similarly, the lower bounds in those proofs use only top-1 packages with one item. Thus these lower bounds are still valid here. It remains to show that for items, (a) FRP(CQ) is $\mathsf{FP}^{\mathsf{NP}}$-hard and (b) MBP(CQ) is DP-hard.

(a) FRP(CQ). We show that FRP(CQ) is $\mathsf{FP}^{\mathsf{NP}}$-hard by reduction from MAX-WEIGHT SAT (see the proof of Theorem 4.3 for the statement of SAT-UNSAT). Consider an instance ($\mathcal{C}$, $\{w_1, \ldots, w_r\}$) of MAX-WEIGHT SAT, where $\mathcal{C}$ is a set of clauses $\{C_1, \ldots, C_r\}$ that are defined over variables in set $X = \{x_1, \ldots, x_m\}$, and for each $i \in [1, r]$, w_i is an integer weight associated with clause C_i. Given such an instance, we define a database D that consists of a single relation I_{01} as shown in Figure A.1, specified by schema $R_{01}(X)$, a query Q as a Cartesian product of relation R_{01} to generate all truth assignments of X variables, and set $k = 1$. Furthermore, for each tuple t in $Q(D)$, we define its rating $f(t)$ as the sum of weights of clauses in $\mathcal{C}$ that are true under the truth assignment encoded by t.

By the definition of $f()$, one can readily verify that for any tuple $t \in Q(D)$, $\{t\}$ is a top-1 item selection for (Q, D, f) if and only if t encodes a truth assignment of X variables that satisfies a set of clauses with the largest total weight. Thus it is a reduction.

(b) MBP(CQ). We show that MBP(CQ) is DP-hard by reduction from SAT-UNSAT (see the proof of Theorem 4.2, CQ case, for the statement of SAT-UNSAT). Given an instance (φ_1, φ_2) defined over $X = \{x_1, \ldots, x_m\}$ and $Y = \{y_1, \ldots, y_n\}$, respectively, we define a database D that consists of a single relation I_{01} as shown in Figure A.1, specified by schema $R_{01}(X)$, a CQ query Q as a Cartesian product of relation R_{01} to generate all truth assignments of $X \cup Y$ variables, and take $k = 1$. Furthermore, for any tuple $t \in Q(D)$, we define (i) $f(t) = 1$ if the truth assignment μ_X of X variables encoded by t makes φ_1 true, while the truth assignment μ_Y of Y variables also encoded by t makes φ_2 false; and (ii) for any other tuple $t' \in Q(D)$, we define $f(t) = 2$. Finally, we set $B = 1$.

We next show that φ_1 is satisfiable and φ_2 is not satisfiable if and only if B is the maximum bound for $(Q, D, f, k = 1)$. By the definition of $f()$, φ_1 is satisfiable and φ_2 is not satisfiable if and only if there exists a tuple $t \in Q(D)$ such that $f(t) = 1$, and moreover, there exists no tuple $t' \in Q(D)$ such that $f(t') > 1$. Obviously, the latter holds if and only if $B = 1$ is the maximum bound for $(Q, D, f, k = 1)$.

Data complexity. Obviously, the algorithms developed for Corollary 6.1 suffice for item selections when Q is fixed. As a result, the upper bounds given there carry over.

This completes the proof of Theorem 6.4. □

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[2] Gediminas Adomavicius and Alexander Tuzhilin. Multidimensional recommender systems: A data warehousing approach. In *Proc. of 2nf Int. Workshop Electronic Commerce (WELCOM)*, volume 2232 of *Lecture Notes in Computer Science*, pages 180–192. Springer, 2001.

[3] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.*, 17(6):734–749, 2005.

[4] Sihem Amer-Yahia. Recommendation projects at Yahoo! *IEEE Data Eng. Bull.*, 34(2):69–77, 2011.

[5] Sihem Amer-Yahia, Senjuti Basu Roy, Ashish Chawla, Gautam Das, and Cong Yu. Group recommendation: Semantics and efficiency. *Proc. of the VLDB Endowment*, 2(1):754–765, 2009.

[6] Albert Angel, Surajit Chaudhuri, Gautam Das, and Nick Koudas. Ranking objects based on relationships and fixed associations. In *Proc. of 12th Int. Conf. on Extending Database Technology (EDBT)*, volume 360 of *ACM Int. Conf. Proceeding Series*, pages 910–921. ACM, 2009.

[7] Alexander Brodsky, Sylvia Henshaw, and Jon Whittle. CARD: A decision-guidance framework and application for recommending composite alternatives. In *Proc. ACM Conf. on Recommender Systems (RecSys)*, pages 171–178. ACM, 2008.

[8] Surajit Chaudhuri and Moshe Y. Vardi. On the equivalence of recursive and nonrecursive Datalog programs. *J. Comput. Syst. Sci.*, 54(1):61–78, 1997.

[9] Sara Cohen and Yehoshua Sagiv. An incremental algorithm for computing ranked full disjunctions. *J. Comput. Syst. Sci.*, 73(4):648–668, 2007.

[10] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv*, 33(3):374–425, 2001.

[11] Ting Deng, Wenfei Fan, and Floris Geerts. The complexity of package recommendation problems. In *Proc. 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 261–272. ACM, 2012.

[12] Arnaud Durand, Miki Hermann, and Phokion G. Kolaitis. Subtractive reductions and complete problems for counting complexity classes. *Theor. Comput. Sci.*, 340(3):496–513, 2005.

[13] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[14] Konstantin Golenberg, Benny Kimelfeld, and Yehoshua Sagiv. Optimizing and parallelizing ranked enumeration. *Proc. of the VLDB Endowment*, 4(11):1028–1039, 2011.

[15] Lane A. Hemaspaandra and Heribert Vollmer. The satanic notations: counting classes beyond #P and other definitional adventures. *SIGACT News*, 26(1):2–13, 1995.

[16] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4):11:1–11:58, 2008.

[17] Georgia Koutrika, Benjamin Bercovitz, and Hector Garcia-Molina. FlexRecs: expressing and combining flexible recommendations. In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD)*, pages 745–758. ACM, 2009.

[18] Mark W. Krentel. Generalizations of Opt P to the polynomial hierarchy. *Theor. Comput. Sci.*, 97(2):183–198, 1992.

[19] Richard E. Ladner. Polynomial space counting problems. *SIAM J. Comput.*, 18(6):1087–1097, 1989.

[20] Theodoros Lappas, Kun Liu, and Evimaria Terzi. Finding a team of experts in social networks. In *Proc. 15th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 467–476. ACM, 2009.

[21] Chengkai Li, Mohamed A. Soliman, Kevin Chen-Chuan Chang, and Ihab F. Ilyas. RankSQL: Supporting ranking queries in relational database management systems. In *Proc. 31st Int. Conf. on Very Large Data Bases (VLDB)*, pages 1342–1345. ACM, 2005.

[22] Apostol Natsev, Yuan-Chi Chang, John R. Smith, Chung-Sheng Li, and Jeffrey Scott Vitter. Supporting incremental join queries on ranked inputs. In *Proc. 27th Inter. Conf. on Very Large Data Bases (VLDB)*, pages 281–290. Morgan Kaufmann, 2001.

[23] Christos H Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[24] Aditya G. Parameswaran, Hector Garcia-Molina, and Jeffrey D. Ullman. Evaluating, combining and generalizing recommendations with prerequisites. In *Proc.19th ACM Conf. on Information and Knowledge Management (CIKM)*, pages 919–928. ACM, 2010.

[25] Aditya G. Parameswaran, Petros Venetis, and Hector Garcia-Molina. Recommendation systems with complex constraints: A course recommendation perspective. *ACM Trans. Inf. Syst.*, 29(4), 2011.

[26] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.

[27] Karl Schnaitter and Neoklis Polyzotis. Evaluating rank joins with optimal cost. In *Proc. 27th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 43–52. ACM, 2008.

[28] Kostas Stefanidis, Georgia Koutrika, and Evaggelia Pitoura. A survey on representation, composition and application of preferences in database systems. *ACM Trans. Database Syst.*, 36(3), 2011.

[29] Larry J. Stockmeyer. The polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1):1–22, 1976.

[30] Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8(2):189 – 201, 1979.

[31] M. Y. Vardi. The complexity of relational query languages. In *Proc. 14th ACM Symposium on Theory of Computing (STOC)*, pages 137–146. ACM, 1982.

[32] Sergei G. Vorobyov and Andrei Voronkov. Complexity of nonrecursive logic programs with complex values. In *Proc. 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 244–253. ACM, 1998.

[33] Michael Wooldridge and Paul E. Dunne. On the computational complexity of qualitative coalitional games. *Artif. Intell.*, 158(1):27–73, 2004.

[34] Min Xie, Laks V. S. Lakshmanan, and Peter T. Wood. Breaking out of the box of recommendations: from items to packages. In *Proc. ACM Conf. on Recommender Systems (RecSys)*, pages 151–158. ACM, 2010.